

Android程序设计

◎ 吕云翔 杨婧 谢文彬 编著





清華大學出版社



Android程序设计

◎ 吕云翔 杨婧 谢文彬 编著

清华大学出版社
北京

内 容 简 介

本书介绍了 Android 应用程序设计的主要思想和方法。首先从 Android 的历史着手，使读者对 Android 这一开源系统的特点有基本的了解；然后深入讲解 Android 的系统架构，以避免读者对此系统只知其然不知其所以然。本书以 Android Studio 为开发工具，因此对该开发环境也做了详细介绍。

在对 Android 有了必要的认知后，本书由浅入深地介绍了 Android 项目的创建和目录结构。对 Android 四大组件、UI、数据持久化和网络编程等主要知识，本书从理论和实践两方面进行了全面的讲解，力求能探究到 Android 设计者的最初想法。

本书可以作为高等院校及各类培训机构 Android 系统课程的教材，也可以作为学习 Android 程序设计人员的自学用书。

图书在版编目（CIP）数据

Android 程序设计/吕云翔等编著. —北京：清华大学出版社，2018
（21 世纪高等学校计算机专业实用规划教材）
ISBN 978-7-302-47726-6

Ⅰ. ①A…　Ⅱ. ①吕…　Ⅲ. ①移动终端－应用程序－程序设计　Ⅳ. ①TN929.53

中国版本图书馆 CIP 数据核字（2017）第 165960 号

责任编辑：魏江江　李　晔
封面设计：刘　键
责任校对：时翠兰
责任印制：刘海龙

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn, http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 装 者：三河市铭诚印务有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：**14.75　　**字　　数：**265 千字
版　　次：2018 年 3 月第 1 版　　**印　　次：**2018 年 3 月第 1 次印刷
印　　数：1～2000
定　　价：39.00 元

产品编号：072601-01

出版说明

随着我国改革开放的进一步深化，高等教育也得到了快速发展，各地高校紧密结合地方经济建设发展需要，科学运用市场调节机制，加大了使用信息科学等现代科学技术提升、改造传统学科专业的投入力度，通过教育改革合理调整和配置了教育资源，优化了传统学科专业，积极为地方经济建设输送人才，为我国经济社会的快速、健康和可持续发展以及高等教育自身的改革发展做出了巨大贡献。但是，高等教育质量还需要进一步提高以适应经济社会发展的需要，不少高校的专业设置和结构不尽合理，教师队伍整体素质亟待提高，人才培养模式、教学内容和方法需要进一步转变，学生的实践能力和创新精神亟待加强。

教育部一直十分重视高等教育质量工作。2007 年 1 月，教育部下发了《关于实施高等学校本科教学质量与教学改革工程的意见》，计划实施“高等学校本科教学质量与教学改革工程（简称‘质量工程’）”，通过专业结构调整、课程教材建设、实践教学改革、教学团队建设等多项内容，进一步深化高等学校教学改革，提高人才培养的能力和水平，更好地满足经济社会发展对高素质人才的需要。在贯彻和落实教育部“质量工程”的过程中，各地高校发挥师资力量强、办学经验丰富、教学资源充裕等优势，对其特色专业及特色课程（群）加以规划、整理和总结，更新教学内容、改革课程体系，建设了一大批内容新、体系新、方法新、手段新的特色课程。在此基础上，经教育部相关教学指导委员会专家的指导和建议，清华大学出版社在多个领域精选各高校的特色课程，分别规划出版系列教材，以配合“质量工程”的实施，满足各高校教学质量和教学改革的需要。

本系列教材立足于计算机专业课程领域，以专业基础课为主、专业课为辅，横向满足高校多层次教学的需要。在规划过程中体现了如下一些基本原则和特点。

（1）反映计算机学科的最新发展，总结近年来计算机专业教学的最新成果。内容先进，充分吸收国外先进成果和理念。

（2）反映教学需要，促进教学发展。教材要适应多样化的教学需要，正确把握教学内容和课程体系的改革方向，融合先进的教学思想、方法和手段，体现科学性、先进性和系统性，强调对学生实践能力的培养，为学生知识、能力、素质

协调发展创造条件。

（3）实施精品战略，突出重点，保证质量。规划教材把重点放在公共基础课和专业基础课的教材建设上；特别注意选择并安排一部分原来基础比较好的优秀教材或讲义修订再版，逐步形成精品教材；提倡并鼓励编写体现教学质量和教学改革成果的教材。

（4）主张一纲多本，合理配套。专业基础课和专业课教材配套，同一门课程有针对不同层次、面向不同应用的多本具有各自内容特点的教材。处理好教材统一性与多样化，基本教材与辅助教材、教学参考书，文字教材与软件教材的关系，实现教材系列资源配套。

（5）依靠专家，择优选用。在制定教材规划时要依靠各课程专家在调查研究本课程教材建设现状的基础上提出规划选题。在落实主编人选时，要引入竞争机制，通过申报、评审确定主题。书稿完成后要认真实行审稿程序，确保出书质量。

繁荣教材出版事业，提高教材质量的关键是教师。建立一支高水平教材编写梯队才能保证教材的编写质量和建设力度，希望有志于教材建设的教师能够加入到我们的编写队伍中来。

21 世纪高等学校计算机专业实用规划教材

联系人：魏江江 weijj@tup.tsinghua.edu.cn

前　言

“这是最好的时代，这是最坏的时代”。

对如今的安卓开发界而言，两百年前，狄更斯说的这句话颇有道理。

这是最好的时代——互联网经济高速发展。雷军的一句“站在风口上，猪也会飞”，话粗理不粗，小米的成功也佐证了这一点。作为移动互联网最主要的载体——智能手机如火山岩浆般喷涌，而安卓手机自然是当中最强势的一流。Statista 的统计数据显示，2016 年第二季度全球手机出货量中，安卓手机占有 86.2%的份额。此番强势表现，对众多的安卓开发者，无疑是最好的时代。

然而，这也是最坏的时代。开放性的发展造成安卓碎片化问题严重。2016 年 3 月，Google 正式发布 Android 7.0。令人尴尬的是，调查显示，截至发布之日，Android 6.0 的市场普及率只有 2.3%，更不要说各个厂商安卓手机的硬件、系统都有着诸多差别。因此，开发者不得不花费大量时间适配不同机型，初学者面临这些问题时，往往不知所措；而市面上多数安卓教材仍沿用过时的理论，基于古老的安卓 4.X，甚至还在使用着官方目前已由 Android Studio 代替的 Eclipse 和已经停止更新的 ADT。

本书旨在更好地解决上述问题，帮助初学者更加高效地接触、了解和熟悉安卓开发。在参阅了许多大同小异的相关书籍后，我们力求能直击安卓的本质，以清晰合理的逻辑，让初学者明白安卓设计的初衷，以设计出高效而不失优雅的安卓程序。对比其他安卓教材，本书具有以下优点：

目标针对性强。本书针对国内计算机、软件相关专业已先修 Java 程序设计课程的学生，旨在为具备良好 Java 编程能力的学生提供一本能够快速熟悉 Android 平台的教材，熟练掌握 Android 开发过程中必备的基础知识，为今后的课程学习和工作打下坚实的基础。

内容与时俱进。计算机学科发展异常迅速，内容更新很快。作为教材，一方面要反映本领域基础性、普遍性的知识，保持内容的相对稳定性；另一方面，也需要不断跟踪科技的发展，本书坚持使用最新的 Android 版本和 2013 年 Google 新推出的 Android Studio 作为开发环境；重点介绍使用新技术的案例，避免使用

即将淘汰的设计方法。

结构合理，习题精要。本书体系结构严谨，概念清晰，内容由浅入深，符合学生的认知规律，易学易懂，且配有许多难度适中、逻辑合理、适于初学者和进阶者开拓思路、深入了解 Android 基础理论和开发技巧的习题以及切合实际的参考答案和章末要点总结，适合教学和自学，是学生掌握 Android 开发的必备书目。

理论结合实践。本书用实例讲授知识点，不局限于枯燥的理论介绍。与许多课程的规律类似，实践对于 Android 学习而言也是强化和提升学习效果的必由之途，否则无异于“入宝山而空返”。读者通过将书中代码手敲一遍或仿照书中实例自己编写小型应用进行练习，可切实强化编码能力，提高软件分析设计能力，真正领悟学习程序设计语言的真谛。

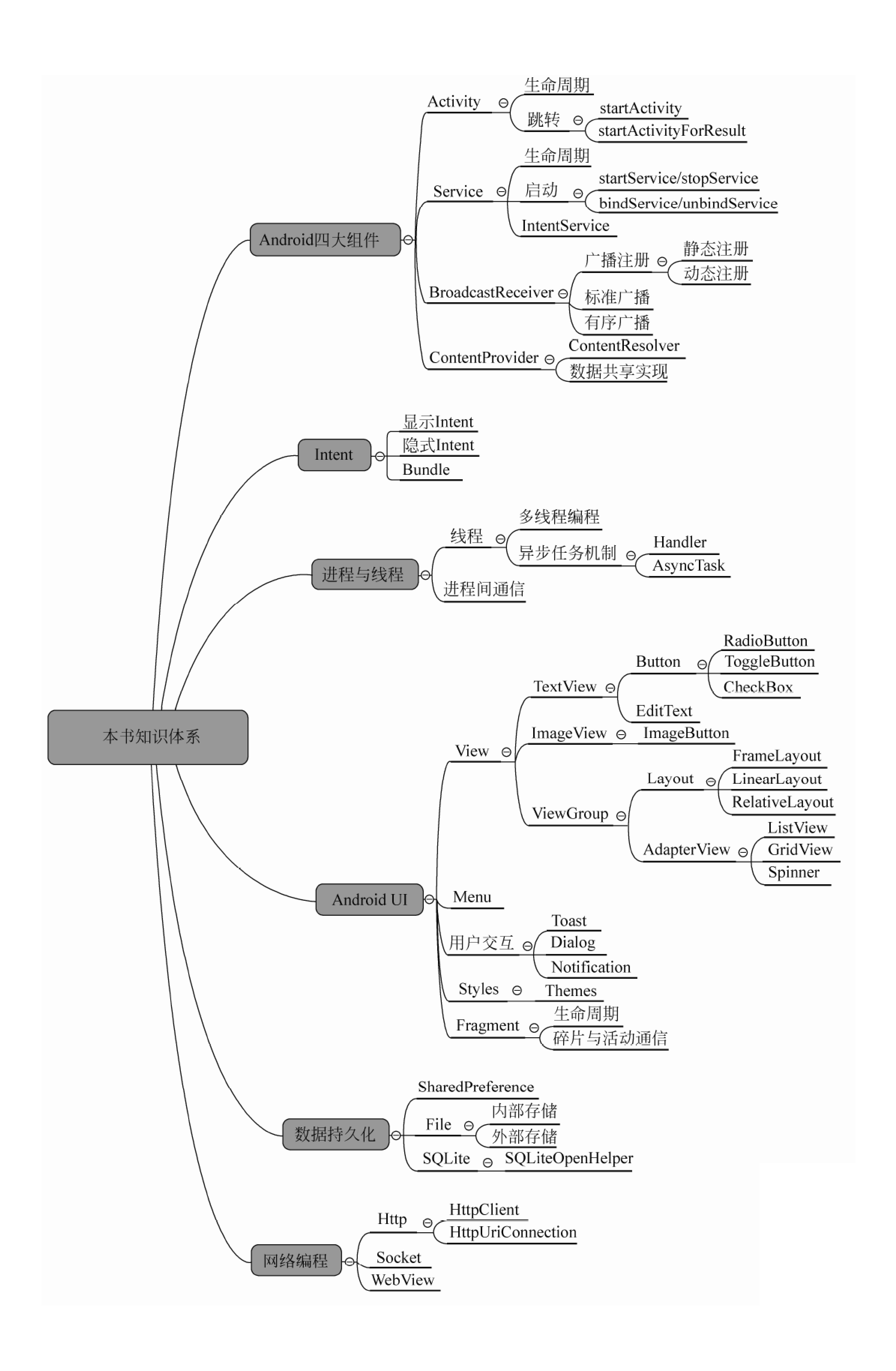
着眼整体认识，体现特色内容。本书注重系统思维，首先展现 Android 基础知识体系的整体框架，然后深入细节，便于读者在脑海中清晰地构建知识网络，实现融会贯通。在具体内容上，力求突出 Android 开发理论中最精华的部分，避免面面俱到、缺少重点，同时增加一些实际开发中可能会用到的高深知识和 Android 中的特色功能，以供读者进一步深入学习。

本书的作者为吕云翔、杨婧、谢文彬，曾洪立、吕彼佳、姜彦华参与了素材整理及配套资源制作。

由于我们的水平和能力有限，本书难免有疏漏之处。恳请各位同人和广大读者给予批评指正，也希望各位能将实践过程中的经验和心得与我们交流（yunxianglu@hotmail.com）。

编　者

2017 年 12 月于北航软件学院


本书知识体系
Android四大组件
Activity
生命周期
跳转
startActivity
startActivityForResult
Service
生命周期
启动
startService/stopService
bindService/unbindService
IntentService
BroadcastReceiver
广播注册
静态注册
动态注册
标准广播
有序广播
ContentProvider
ContentResolver
数据共享实现
Intent
显示Intent
隐式Intent
Bundle
进程与线程
线程
多线程编程
异步任务机制
Handler
AsyncTask
进程间通信
Android UI
View
TextView
Button
RadioButton
ToggleButton
CheckBox
EditText
ImageView
ImageButton
ViewGroup
Layout
FrameLayout
LinearLayout
RelativeLayout
AdapterView
ListView
GridView
Spinner
Menu
用户交互
Toast
Dialog
Notification
Styles
Themes
Fragment
生命周期
碎片与活动通信
数据持久化
SharedPreference
File
内部存储
外部存储
SQLite
SQLiteOpenHelper
网络编程
Http
HttpClient
HttpUriConnection
Socket
WebView

目　　录

第 1 章　Android 概述

1.1　了解 Android

科技高速发展，芯片的集成度越来越高，摩尔定律百试不爽。所有的一切仿佛都在默默发酵，为了一场即将到来的革命做准备。果不其然，21 世纪的第一个十年，这场革命就掀起了 PC 发展的高潮。这个高潮中最大的赢家莫过于微软（Microsoft）和英特尔（Intel）公司。然而就在人们以为这两家公司将一劳永逸的时候，革命还未停止，又一场高潮到来——移动设备和移动互联网迅猛发展。一时之间，移动设备的龙头——智能手机风光无限，竟盖过了 PC 的风头，甚至于当时世界上最大的 PC 设备制造商——惠普公司一度传闻要出售 PC 业务。在这场革命中，以桌面系统占领市场的微软公司显得有点不适应，步履蹒跚，反倒是以搜索业务起家的谷歌（Google）公司和以 PC 业务发家的苹果（Apple）公司玩得风生水起。究其原因，自然是这两家公司旗下的移动设备操作系统各有其独到之处。苹果公司的 iOS 系统独辟蹊径，虽封闭却不失人性化。而 Google 公司的 Android（安卓）系统则以开放和可自由定制与更改而著称，占据着移动设备操作系统的大部分市场份额。那么 Android 究竟是什么？它是如何壮大的？又是怎么工作的呢？本章为你一一揭晓答案。

1.1.1　Android 起源与发展

Android 系统是一款基于 Linux 的自由及开放源代码的智能操作系统，主要用于智能手机和平板电脑等移动设备，由 Google 公司和开放手持设备联盟（Open Handset Alliance，OHA）领导及开发。

Android 一词的本义为“机器人”，第一次出现是在法国作家维里耶德利尔 •亚当在 1886 年发表的科幻小说《未来夏娃》（*L'ève future*）中。该书的男主角为报答救命恩人，为其制造了一个女性机器人，名唤 Hadaly，而此类仿人机器人在书中被称为 Android。这也就不难理解 Android 系统的最终设计目标——使移动设备

更加具有人工智能，更贴近人类生活。

Android 系统最初由安迪 • 鲁宾（Andy Rubin，被誉为 Android 之父）领导的 Android 公司开发，主要支持手机设备。2005 年 8 月 17 日，Google 公司低调收购了这家成立仅 22 个月的高科技企业及其团队。安迪 • 鲁宾成为 Google 公司工程部副总裁，继续负责 Android 项目。2007 年 11 月 5 日，Google 公司正式向外界展示了 Android 智能操作系统，并且在这天宣布建立一个全球性的联盟组织——开放手持设备联盟，以共同研发改良的 Android 系统。该组织由 34 家手机制造商、软件开发商、电信运营商以及芯片制造商共同组成，这一联盟将支持 Google 公司发布的手机操作系统以及应用软件。随后 Google 公司以 Apache 免费开源许可证的授权方式，发布了 Android 的源代码。自此，Android 系统正式问世，并快速走上了正轨。

2008 年 9 月 23 日，Google、HTC（宏达国际电子股份有限公司）、T-Mobile（德国电信的子公司）共同发布了全球第一部搭载 Android 操作系统的智能手机——T-Mobile G1（又称 HTC Dream），如图 1-1 所示。



图 1-1　第一部 Android 手机——T-Mobile G1

从 2008 年第一台 Android 智能手机发布到 2016 年，只有 8 年光阴。而这短短 8 年时间，Google 将曾经的智能手机操作系统霸主 Symbian（塞班）拉下台，

又在与 iOS、Windows Phone（微软公司旗下的手机操作系统）的斗争中一路高歌猛进。时至今日，Android 已成为市场份额最大的智能手机操作系统。

1.1.2 开放手持设备联盟

Android 系统的成功自然不是个偶然事件。众多组织和个人在此过程中做出了不可磨灭的贡献，其中开放手持设备联盟的领导尤为重要。

开放手持设备联盟是美国 Google 公司于 2007 年 11 月 5 日宣布组建的一个全球性的联盟组织。这一联盟将会支持 Google 发布的 Android 系统和其应用软件，共同开发 Android 系统。该联盟旨在统一手持设备的开放准则，研发用于移动设备的新技术，以减少移动设备开发与推广成本，同时建立移动通信领域新的协作环境，借助标准化、开放式的移动软件平台，在移动产业内形成一个开放式的生态系统。

开放手持设备联盟包括移动运营商、半导体芯片商、手机硬件制造商和软件开发商几类。成立之初，联盟成员数为 34 家。伴随着 Android 系统的高速发展，目前，联盟成员数量已经达到了 84 家。

首批 34 名成员中的电信运营商有 KDDT（日本）、NTT(日本)、Sprint（美国）、T-Mobile（德国）、中国移动、Telecom Italia（意大利）和 Telefónica（西班牙）。

半导体芯片商包括 Audience（美国）、Broadcom（美国）、CSR（英国）、Intel（美国）、Marvell（美国）、NVIDIA（美国）、Qualcomm（美国）、Synaptics（美国）和 Texas（美国）。

手机硬件制造商包括 HTC（中国台湾）、LG（韩国）、Sony（日本）、Motorola（美国，现已被联想收购）和 Samsung（韩国）。

软件厂商有 Ascender Corp（美国）、eBay(美国)、Google（美国）、LivingImage（日本）、Myriad（瑞士）、Nuance Communications（美国）、PacketVideo（美国）、SkyPoP（美国）SONiVOX（美国）和 Wind River Systems（美国）。

此外还包括 Flex Comix（日本）、Nexus Telecom（瑞士）和 The Astonishing Tribe（瑞典）。

1.1.3 Android 市场占有率

随着 Windows Phone 以及其他更为小众的手机操作系统（如 BlackBerry OS、Ubuntu 手机版等）的没落，如今全球范围内的手机智能操作系统市场，愈发呈现出两极分化的态势——Android 系统与 iOS 系统两者各领风骚。其中，iOS 凭借

iPhone 牢牢占据着高端市场，而 Android 的整体表现则更为均衡。市场研究机构 Kantar Worldpanel 最新发布的报告称，截至 2016 年第二季度，Android 操作系统在全球所占份额已经逼近 80%，其在中国市场的占有率甚至高达 85%，如图 1-2 所示。

Smartphone OS Sales Share (%)

Germany	3 me Jul 15	3 me Jul 16	% pt. Change
Android	73.7	79.8	6.1
iOS	15.2	15.2	0.0
Windows	10.1	4.8	-5.3
Other	1	0.1	-0.9
GB	**3 me Jul 15**	**3 me Jul 16**	**% pt. Change**
Android	54.4	57.3	2.9
iOS	32.8	38.0	5.2
Windows	11.9	4.3	-7.6
Other	1	0.4	-0.6
France	**3 me Jul 15**	**3 me Jul 16**	**% pt. Change**
Android	69.6	75.6	6.0
iOS	17	18.8	1.8
Windows	12.6	4.9	-7.7
Other	0.7	0.7	0.0
Italy	**3 me Jul 15**	**3 me Jul 16**	**% pt. Change**
Android	72.9	82.5	9.6
iOS	11.4	12.7	1.3
Windows	14.3	4.7	-9.6
Other	1.4	0.1	-1.3
Spain	**3 me Jul 15**	**3 me Jul 16**	**% pt. Change**
Android	89.4	90	0.6
iOS	6.6	9.2	2.6
Windows	4	0.6	-3.4
Other	0	0.1	0.1

USA	3 me Jul 15	3 me Jul 16	% pt. Change
Android	65.6	65	-0.6
iOS	30.1	31.3	1.2
Windows	3.8	2.4	-1.4
Other	0.5	1.3	0.8
China	**3 me Jul 15**	**3 me Jul 16**	**% pt. Change**
Android	79.4	85.0	5.6
iOS	18.7	14.3	-4.4
Windows	1.5	0.2	-1.3
Other	0.4	0.5	0.1
Australia	**3 me Jul 15**	**3 me Jul 16**	**% pt. Change**
Android	56.1	60.2	4.1
iOS	34.9	35.2	0.3
Windows	6.9	2.9	-4.0
Other	2.1	1.7	-0.4
Japan	**3 me Jul 15**	**3 me Jul 16**	**% pt. Change**
Android	62.9	64.6	1.7
iOS	35.1	34.1	-1.0
Windows	0.1	0.6	0.5
Other	1.8	0.6	-1.2
EU5	**3 me Jul 15**	**3 me Jul 16**	**% pt. Change**
Android	71.0	77.0	6.1
iOS	17.2	18.4	1.2
Windows	10.9	4.2	-6.7
Other	0.9	0.3	-0.6

图 1-2　各主要市场的手机操作市场份额

1.2　Android 版本

Android 系统发布至今，经历过多次重大修改，版本迭代，基本保持着一年一次大升级。本节将简要讲述 Android 系统的重要版本信息和主要功能更新以及各版本的市场份额占比，以供读者在 Android 应用时选择合适的 SDK 平台（SDK

Platform）版本。

1.2.1 Android 版本简介

1．Android 1.0 版本

发布于 2008 年 9 月，是 Android 第一个稳定版本。官方提供的 SDK 中包含 Android 模拟器以及丰富的开发工具。

2．Android 1.1 版本

发布于 2009 年 2 月。修复了 1.0 版本存在的诸多问题，如设备休眠时的稳定性、邮件冻结问题、POP3 连接失败问题等。同时增加新特性，例如保存彩信附件等。

3．Android 1.5 版本

代号为 Cupcake（纸杯蛋糕），发布于 2009 年 4 月。从这一版本开始，Google 正式采用甜品名作为 Android 版本代号。Cupcake 大幅提高了性能表现，主要更新如下：

- 支持立体声蓝牙耳机，同时改善自动配对性能；
- 更新基于 WebKit 的浏览器；
- 提高 GPS 性能；
- 提供屏幕虚拟键盘；
- 自动旋转；
- 来电照片显示等。

4．Android 1.6 版本

代号为 Donut（甜甜圈），发布于 2009 年 9 月。主要更新如下：

- 重新设计的 Android Market 手势；
- 支持 CDMA 网络；
- 文字转语音系统（Text-to-Speech）；
- 快速搜索框；
- 全新的拍照接口；
- 查看应用程序耗电；
- 支持虚拟私人网络（VPN）；
- 支持更多的屏幕分辨率；
- 支持 OpenCore2 媒体引擎；
- 新增面向视觉或听觉困难人群的易用性插件等。

5. Android 2.0/2.1 版本

代号为 Éclair（松饼），发布于 2009 年 10 月。主要更新如下：

- 优化硬件速度；增加 Car Home 程序；
- 改良的更为人性化的用户界面；
- 支持更多的屏幕分辨率；新的浏览器的用户接口和支持 HTML5；
- 新的联系人名单；
- 更好的白色/黑色背景比率；
- 改进 Google Maps3.1.2；
- 支持 Microsoft Exchange；
- 支持内置相机闪光灯；支持数码变焦；
- 改进的虚拟键盘；
- 支持蓝牙 2.1；
- 支持动态桌面的设计等。

6. Android 2.2/2.2.1 版本

代号为 Froyo（冻酸奶），发布时间为 2010 年 5 月。主要更新如下：

- 整体性能大幅度提升；
- 3G 网络共享功能；
- 支持 Flash；
- App2sd 功能；
- 全新的软件商店；
- 更多的 Web 应用 API 接口的开发等。

7. Android 2.3 版本

代号为 Gingerbread（姜饼），发布于 2010 年 12 月 7 日。主要更新如下：

- 增加了新的垃圾回收和优化处理事件；
- 新的管理窗口和生命周期的框架；
- 支持 VP8 和 WebM 视频格式；
- 提供 AAC 和 AMR 宽频编码；
- 提供了新的音频效果器；
- 支持前置摄像头、SIP/VOIP 和 NFC（近场通信）；
- 改进的电源管理系统；采用新的应用管理方式等。

8. Android 3.0 版本

代号为 Honeycomb（蜂巢），发布于 2011 年 2 月 2 日。该版本只支持平板电

脑。其针对平板电脑做了大量优化，但由于增加了开发者工作量和用户体验一般，该版本很快就被市场抛弃了。

9. Android 4.0 版本

代号为 Ice Cream Sandwich（冰激凌三明治），于 2011 年 10 月 19 日发布。主要更新如下：

- 全新的 UI；
- 全新的 Chrome Lite 浏览器；
- 截图功能；
- 更强大的图片编辑功能；
- Gmail 加入手势、离线搜索功能；
- 新功能 People——以联系人照片为核心，界面偏重滑动而非点击，集成了 Twitter、LinkedIn、Google+等通信工具；
- 新增流量管理工具，可具体查看每个应用产生的流量，限制使用流量，到达设置标准后自动断开网络等。

10. Android 4.1/4.2 版本

代号为 Jelly Bean（果冻豆），两个版本分别发布于 2012 年 6 月和 10 月。主要更新如下：

- 极大地提高了系统体验；
- 特效动画的帧速提高至 60fps，增加了三倍缓冲；
- 全新搜索——搜索将会带来全新的 UI、智能语音搜索和 Google Now 三项新功能；
- 桌面插件自动调整大小；
- 加强无障碍操作；
- 语言和输入法扩展；
- Photo Sphere 全景拍照功能；
- Daydream 屏幕保护程序；
- 支持 Miracast 无线显示共享功能；
- 更加智能的 Google Now。

11. Android 4.4 版本

代号为 KitKat（奇巧巧克力），于 2013 年 11 月 01 日正式发布。新的 4.4 系统更多地整合了自家服务，力求防止 Android 系统继续碎片化、分散化。该版本

支持两种编译方式——默认的 Dalvik 和 ART，对内存做了极为优秀的优化，甚至可以在内存为 512MB 的旧款手机上顺畅运行。此外还增加了低功率音频和定位模式，有效提高了移动设备续航时间。增加了新的蓝牙配置文件，增强了红外线兼容性。

12．Android 5.0 版本

代号为 Lollipop（棒棒糖），于 2014 年 11 月正式发布。Android 5.0 系统使用一种新的 Material Design 设计风格，以统一 Android 设备的外观和使用体验。默认的编译模式也由 ART 取代了 Dalvik，从而使性能提高了 4 倍。同时该版本支持更多的传感器，并且支持 64 位处理器。此外，其新增了自动内容加密功能和多人设备分享功能，以及全新的更为人性化的锁屏界面设计。

13．Android 6.0 版本

代号为 Marshmallow（棉花糖），正式发布于 2015 年 5 月。新系统的整体设计风格依然保持扁平化的 Material Design 风格，增加了大量漂亮流畅的动画；在软件体验与运行性能上进行了大幅度的优化；提供了原生的权限管理工具；相机新增专业模式，并且支持 RAW 格式照片。

14．Android 7.0 版本

代号为 Nougat（牛轧糖）官方又称为 Android N，开发者预览版发布于 2016 年 5 月。该版本采用了先进的图形处理 Vulkan 系统，能有效减少对 CPU 的占用。与此同时，Android N 加入了 JIT 编译器，安装程序快了 75%，所占空间减少了 50%。此外，Android N 原声支持分屏多任务、Data Saver（流量管理）、号码拦截、Java8 语言支持、画中画功能。

1.2.2 Android 各版本市场份额

2016 年 8 月 3 日，Google 官方给出最新的 Android 各版本装机率统计（见表 1-1）。统计显示，目前 Android 4.x 仍占据着最大的市场份额，发布一年多的 Android 6.0 所占市场份额只有 15.2%。Android 碎片化问题依旧严重。

表 1-1　2016 年 8 月，Android 各版本装机率

版　　本	代　　号	API	占有率/%
2.2	Froyo	8	0.1
2.3.3～2.3.7	Gingerbread	10	1.7
4.0.3、4.0.4	Ice Cream Sandwich	15	1.6

续表

版　　本	代　　号	API	占有率/%
4.1.x	Jelly Bean	16	6.0
4.2.x		17	8.3
4.3		18	2.4
4.4	KitKat	19	29.2
5.0	Lollipop	21	14.1
5.1		22	21.4
6.0	Marshmallow	23	15.2

1.3　Android 的特征

Android 之所以能从众多的手机操作系统中脱颖而出，自然有其独到的优势：

- 开放。Google 完全公开了 Android 的源代码，而且对其大部分代码都采用 Apache 公开协议。用户可以免费安装使用 Android，移动设备商不需要缴纳系统权利金，有效降低了移动设备的成本。同时，移动设备商可以根据自己的需求改造系统，并且不同于 GPL 授权方式，Apache 协议授权移动设备商有权不公开源代码，从而有效地提高了产品竞争力。
- 应用范围广。除了可应用于智能手机上，Android 还可以用于平板电脑、智能电视、智能手表、电视盒子、MP4 及 PC 等诸多设备，应用范围极广。
- 功能丰富、可拓展性强。Android 广泛支持 GSM、3G 及 4G 的语音和数据业务，支持语音呼叫、SMS、数据存储共享和 IPC 详细机制，提供地理服务 API 函数库、组件复用和内置程序替换的应用程序框架。广泛支持多种视频、图像和音频文件。界面方面提供了丰富的界面控件。在内存和进程管理方面，具有自己的运行时和虚拟机。
- 硬件兼容性强。Android 提供了访问硬件的 API 库函数，使得使用者可以简单方便地调用硬件。并且，所有安装 Android 系统的手机，对硬件的访问方式是一样的，因此，即便将应用程序移植到其他不同型号的手机，也不需要修改硬件访问方法。Android 支持丰富的硬件，包括摄像头、蜂窝网络、GPS、NFC、WiFi、蓝牙、加速度计、触摸屏、多种传感器、红外发射及电源管理等。
- 开发便捷。Google 为开发者提供了强大而丰富的开发工具，全新的 Android Studio 能帮助开发者有效地将注意力放在程序逻辑本身。更为重要的是，

Android 代码开发使用的 Java——一种极为流行并且跨平台的语言，这一举动使得 Android 开发者不需要像 iOS 开发者那样还需要特意新学一门语言。

- 界面与逻辑分离。Android 业务逻辑开发主要使用 Java，而界面设计主要用 XML。XML 是可扩展标记语言，使用方便简单，能帮助界面设计师更为专注界面本身，设计出精美的多设备兼容的 UI。

1.4 Android 系统架构

Android 采用层次化系统架构，各层次分工明确，如图 1-3 所示。由上往下分为 4 个主要功能层，分别为应用程序层（Applications）、应用程序架构层（Application Framework）、系统运行时库层（包括 Libraries 和 Android Runtime）和 Linux 内核层（Linux Kernel）。采用层次化系统架构的好处是充分体现了高内聚、低耦合的特点——本层可以使用下一层的服务，同时为上一层提供服务，而且本层及以下层的变化不会影响上一层的功能，各层各司其职，便于系统开发。

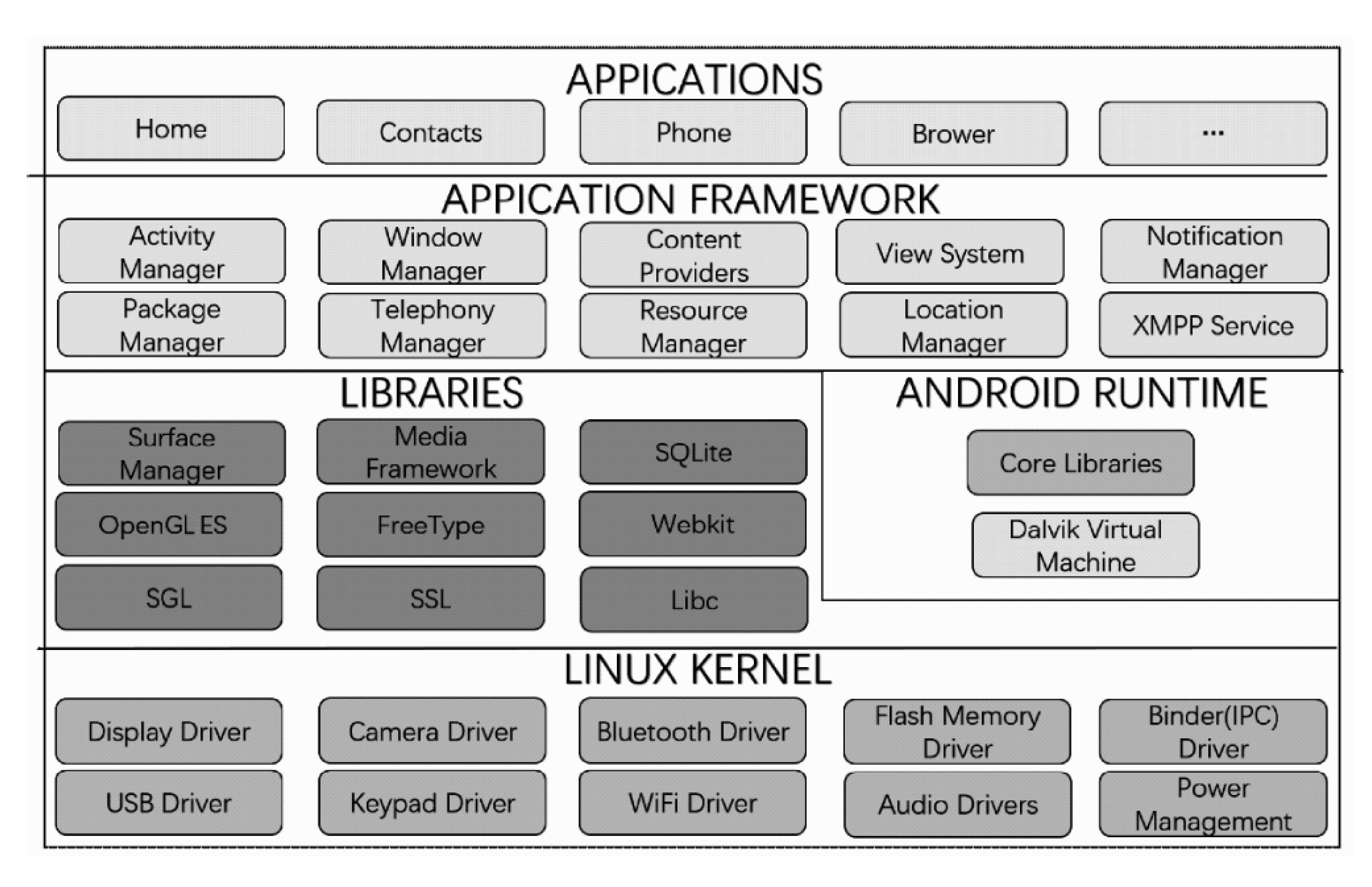


图 1-3 Android 系统架构

1.4.1 应用程序层

Android 系统的应用程序层包含各类与用户直接交互的应用程序，以及由 Java 语言编写的运行于后台的服务程序，例如 SMS 短信、时钟、日历、游戏、地图等

程序，以及开发人员开发的其他应用程序。该层为普通用户与系统直接交互的重要一环，直接决定了系统的用户体验。

1.4.2 应用程序架构层

应用程序架构层提供开发 Android 应用程序所需的诸多类库，使开发人员可以进行快速便捷的应用程序开发，更为方便地重用组件，也可以通过继承实现个性化的扩展，实现诸多功能。具体包含的模块如表 1-2 所示。

表 1-2 应用程序架构层类库

类 库 名 称	功 能
窗口管理器（Window Manager）	管理所有开启的窗口程序
内容提供器（Content Provider）	提供一个应用程序访问另一个应用程序数据的功能，以及实现应用程序之间的数据共享
视图系统（View System）	创建应用程序的基本 UI 组件，包括按钮（buttons）、列表（lists）、文本框（text boxes）、可嵌入的 Web 浏览器等
通知管理器（Notification Manager）	使应用程序可以在状态栏中显示自定义的客户提示信息
包管理器（Package Manager）	对应用程序进行管理，提供的功能包括安装应用程序、卸载应用程序、查询相关权限信息等
资源管理器（Resource Manager）	提供各种非代码资源供应用程序使用，如本地化字符串、图片、音频等
位置管理器（Location Manager）	提供位置服务
电话管理器（Telephony Manager）	管理手机通话状态、获取电话信息、侦听电话状态以及拨打电话
XMPP 服务	Google 在线即时交流软件中一个通用的进程，提供后台推送服务

1.4.3 系统运行时库层

系统运行时库层是应用程序架构层的支撑，为 Android 系统中的各个组件提供服务。系统运行时库层由系统类库（Libraries）和 Android 运行时（Android Runtime）构成。

1．系统类库

系统类库大部分用 C/C++语言编写，能被 Android 系统中不同的组件使用，其所提供的功能通过 Android 应用程序框架为开发者所用。主要的系统类库及说

明如下：

- 系统 C 库（Bionic Libc）。标准 C 系统库的 BSD（Berkeley Software Distribution，UNIX 操作系统的一个分支）衍生，更适合基于嵌入式 Linux 的移动设备。
- 界面管理（Surface Manager）。执行多个应用程序时，管理子系统的显示，同时可以无缝组合多个应用的二维和三维图形层。
- 媒体库（Media Framework）。基于 PacketVideo 的 OpenCore，支持多种常用的音频、视频格式及静态图像文件，所支持的编码格式包括 MPEG4、MP3、H264、AAC、ARM、JPG、PNG 等。
- 3D 库（OpenGL|ES）。基于 OpenGL ES 1.0API 标准实现的 3D 跨平台图形库。
- SQLite。一种强大而轻量级的关系数据库。Android 提供了一系列崭新的 SQLite 数据库 API，以替代传统的资源耗费率极高的 JDBC API。
- FreeType。用于显示位图和矢量字体。
- SGL。基本的 2D 图形引擎。
- Webkit。Web 浏览器引擎，驱动 Android 浏览器和可嵌入的 Web 视图。

除上述主要系统类库之外，Google 还为开发者提供了 Android NDK（Native Development Kit），即 Android 原生库。通过使用 NDK，开发者可以直接调用 Android 系统资源，并采用 C 或 C++语言编写程序的接口。因此，第三方应用程序可以不依赖于 Dalvik 虚拟机进行开发。NDK 提供了一系列从 C 或 C++生成原生代码所需要的工具。利用这些工具，开发者可以方便快捷地快速开发 C 或 C++的动态库，提高应用程序的运行效率。NDK 提供的工具能自动将生成的动态库和 Java 应用程序一起打包成应用程序包文件，即.apk 文件。

2．运行时

Android 运行时包含核心库和 Dalvik 虚拟机两部分。

1）核心库

核心库提供了 Java SE API 的多数功能（最新的 Android N 支持 Java 8），并提供 Android 的核心 API，如 android.os、android.net、android.media 等。

2）Dalvik 虚拟机

Dalvik 虚拟机是基于 Apache 的特殊 Java 虚拟机，并被施以诸多改进以适应低内存、低处理器速度的移动设备环境。Dalvik 虚拟机依赖于 Linux 内核，实现了进程隔离与线程调试管理、安全和异常管理、垃圾回收等重要功能。但 Dalvik

虚拟机并非传统意义上的 Java 虚拟机（JVM）。Dalvik 虚拟机的实现没有遵循 Java 虚拟机的规范，因此两者并不兼容。

Dalvik 虚拟机和标准 Java 虚拟机有以下主要区别：

- Dalvik 虚拟机基于寄存器，而 JVM 基于栈。基于寄存器的设计使得 Dalvik 依赖于具体的 CPU 底层结构，因此硬件通用性较差，但性能比传统 JVM 更有优势。
- Dalvik 虚拟机允许在有限的内存中同时高效地运行多个虚拟机的实例，并且每一个 Dalvik 应用作为一个独立的 Linux 进程执行，都拥有一个独立的 Dalvik 虚拟机实例。这种基于 Linux 的进程机制叫作“沙箱（Sandbox）”，是 Android 安全设计的基础之一。
- Dalvik 虚拟机是从 DEX（Dalvik Executable）格式文件中直接读取指令与数据，然后解释运行的。DEX 文件由传统编译产生的 CLASS 文件，经 dx 工具软件处理后生成。
- DEX 文件可进一步优化，提高运行性能。通常，OEM 的应用程序可以在系统编译后，直接生成优化文件（.ODEX）；第三方的应用程序则可在运行时在缓存中优化与保存，优化后的格式为 DEY（.dey 文件）。

1.4.4 Linux 内核层

Android 系统是基于 Linux 的自由及开放源代码的操作系统，以 Linux 内核为基础，实现进程与内存管理、硬件驱动、电源管理、无线通信等核心功能。青出于蓝而胜于蓝。Linux 内核层作为硬件和软件之间的抽象层，隐藏具体硬件设计，并为上一层提供统一服务。Android 内核对传统 Linux 内核做了诸多改进，增加了诸多功能，以适应移动设备环境的特点。例如，低内存管理器（Low Memory Killer，LMK），匿名共享内存（Ashmem）和轻量级的电源管理（PM）。此外，Android 内核还去除了传统 Linux 的一些不必要功能，例如本地窗口系统等。Android 在继承 Linux 内核安全机制的同时，进一步提升了内存管理以及进程间通信等方面的安全性，还对 Linux 设备驱动进行了增强。下面列出 Android 内核的主要改进：

- Android Binder。基于 OpenBinder 框架，用于提供 Android 平台的进程间通信（InterProcess Communication，IPC）功能。
- Android 电源管理（PM）。基于标准 Linux 电源管理系统的轻量级 Android 电源管理驱动，对嵌入式系统做了有针对性的优化。
- 低内存管理器（Low Memory Killer）。比 Linux 的标准的 OOM（Out Of

Memory）机制更加灵活，可以根据需要杀死进程来释放需要的内存。

- 匿名共享内存（Ashmem）。为进程之间提供共享内存资源，同时为内核提供回收和管理内存的机制。
- 定时器（Android Alarm）。提供了一个定时器，用于把设备从睡眠状态唤醒，并且提供了一个即使在设备睡眠时也会运行的时钟基准。
- USB Gadget 驱动（USB Gadget Driver）。基于标准 Linux USB Gadget 驱动框架的设备驱动。
- 物理内存映射管理（Android PMEM）。向用户空间提供连续的物理内存区域映射。
- 日志（Android Logger）。一个轻量级的日志设备。
- Yaffs2 文件系统。Android 采用大容量的 NAND 闪存作为存储设备，使用 Yaffs2 作为文件系统管理大容量 MTD NAND Flash。Yaffs2 占用内存小，垃圾回收简捷迅速。

1.5　Android 四大组件

Android 应用程序的基本功能模块是组件。组件之间可以互相调用、互相协调，同时又保持自身的独立性。目前，Android 应用程序有四大组件——活动（Activity）、服务（Service）、广播接收器（Broadcast Receiver）和内容提供器（Content Provider）。

1.5.1　Activity

Activity 是 Android 应用程序最基本的组件，与 Windows 系统中的“窗体”颇为相似。Activity 是用户进行交互的可视化界面。一般来说，一个 Activity 通常就是一个单独的屏幕，其上面可以显示一些空间，也可以监听并处理与用户交互所产生的界面时间。

正常情况下，Android 应用程序可以有一个或多个 Activity，但必须有一个默认的 Activity 作为程序打开的入口。系统会自动打开该 Activity，并呈现给用户对应的界面。当用户切换屏幕的时候，其实后台执行的便是对应 Activity 的切换。系统会把旧 Activity 设为暂停状态，并执行新 Activity。这一过程是基于内存中堆栈的数据结构。本书第 4 章会详细介绍 Activity。

1.5.2 Service

大部分操作系统的应用程序都有类似的组件。Android 中的 Service 就类似 Windows 系统中的 Windows Service。Service 是一个长生命周期、没有用户界面、在后台运行的组件。它常被用来执行后台长期任务，例如，后台播放音乐、下载文件、定时刷新数据和闹钟提醒等。

播放音乐时，用户可能在浏览网页，这时候手机屏幕呈现的是网页的 Activity，而音乐播放是在后台中 Service 中运行的。关于 Service 的更多信息，会在本书第 7 章介绍。

1.5.3 Broadcast Receiver

Broadcast Receiver 是用来接收并响应广播的组件。Android 应用程序可以通过它对外部事件进行过滤，只接收感兴趣的外部事件并做出响应。广播接收器虽然没有界面，但它可以通过启动一个 Activity 或 Service 来响应接收到的信息，或者通过 Notification Manager 通知用户。通知有多种方式吸引用户注意力，例如，震动、播放声音或者在状态栏生成持久图标等。一个程序可以有多个广播接收器，所有的广播接收器都要继承自 android.content.BroadcastReceiver。更多 Broadcast Receiver 的知识，请参考本书第 6 章。

1.5.4 Content Provider

Content Provider 是 Android 系统提供的标准共享数据机制。它可以将一个应用程序的指定数据集提供给其他应用程序。这些数据可能存储在文件系统、SQLite 数据库或其他合理方式下。用户可以实现自定义的内容提供器，也可以调用系统自定义的内容提供器。内容提供器为存储、读取、删除数据提供了统一接口，统一了数据访问方式，使得应用程序间可以实现便捷的数据共享。更多关于内容提供器的知识，请参考本书第 8 章。

1.6 Android 程序生命周期

基本所有操作系统都有“程序生命周期”这一概念，因为操作系统必须控制设备资源的合理分配和调度，为此，不得不在需要的时候“孕育”某一程序，然后在必要的时候“扼杀”它。Android 程序生命周期在硬件层面指的是 Android 系统中进程从启动到终结的所有阶段，在软件层面指的则是 Android 程序从启动

到停止的全过程。

一般而言，Android 系统是运行在资源受限的硬件平台上的。在这种情况下，Android 改进的 Linux 内核中的低内存管理器便有大展身手的机会。Android 系统会利用低内存管理器采用主动式的资源管理，即 Android 程序本身不能完全控制自身的生命周期，而是由 Android 系统进行调度和控制。为保证高优先级程序的正常运行，Android 系统可以在无任何警告的状态下终止低优先级程序。尽管如此，为了保证良好的用户体验，Android 系统会尽量不主动终结应用程序，即便其生命周期结束也会暂时保存在内存中，以便再次快速启动。但在特殊情况下——如果内存紧急即可用内存较小，系统会自动清除低优先级的进程。进程按优先级由高到低可分为前台进程（foreground process）、可见进程（visible process）、服务进程（service process）、后台进程（background process）和空进程（empty process）。

（1）前台进程。Android 系统中最重要的进程，指正在与用户交互的进程。可分为 4 种情况：

- 进程中的 Activity 正在与用户交互。
- 进程中的服务被正与用户交互的 Activity 的其他进程调用。
- 进程中的广播接收器（Broadcast Receiver）正在执行 onReceive()函数（详情参考第 6 章）。
- 进程中的服务正在执行生命周期的回调函数，如 onCreate()、onStart()或 onDestroy()（详情参考第 7 章）。

（2）可见进程。指部分程序界面可被用户看见，但不在前台与进程交互，不响应界面事件的进程。比如部分手机有来信预览功能——当接收到新短信时，会在当前主界面弹出一个小对话框，以供用户预览和快速回复。在这种情况下，小对话框所在的进程是前台进程，而当前主界面所在的进程便是可见进程。

（3）服务进程。指包含已启动服务的进程（但不在前台进程的 4 种情况中）。如后台播放音乐的进程。

（4）后台进程。指不包括任何已启动的服务，并且没有任何用户可见的 Activity 的进程。例如，一个只有 Activity 组件的进程，当其他程序完全遮挡了该进程 Activity 界面时，其便可称为后台进程。成为后台进程时间越久的进程，其优先级越低。

（5）空进程。当后台进程被终结，会将所占大部分内存空间释放，该进程便进入空进程，但进程中的 Activity 仍然存在。将进程转换到空进程，而不是直接“杀死”，主要是为了必要时将其快速恢复为前台进程，而无须重新产生 Activity。

在 Android 系统中，进程的优先级取决于进程对应的程序的所有组件中的最高优先级部分。比如一个进程同时有正在与用户进行交互的 Activity 和已启动但未与其他前台进程交互的服务，则该进程是前台进程而不是服务进程。

习 题 1

1．目前市面上流行的智能手机操作系统有哪些？

2．iOS 和 Android 系统各自的主要优势在哪里？

3．Android 是基于哪个操作系统的？Android 和它所基于的系统主要区别在哪里？

4．请简要地阐述安卓的系统架构。

5．要实现一边播放歌曲，一边进行其他操作，需要哪个组件？播放歌曲的进程属于哪种进程？

第 2 章　构建 Android 程序

本章将讲解如何创建一个 Android 应用程序，同时介绍一些 Android 应用程序设计的基础知识，包括 Android 项目的文件目录结构、项目资源的创建使用与本地化以及项目的调试运行。

2.1　Android 项目创建

在创建项目之前，首先请确保已经安装了必要的开发环境。你需要：下载并安装 Android Studio；使用 SDK Manager 下载最新的 SDK tools 和 platforms，具体安装步骤参见附录。

本节介绍如何使用 Android Studio 来创建一个新的项目，当然除此之外也可以使用命令行来创建项目，对于这种方法在此不做详细介绍，可自行查阅相关资料了解。

首先，启动 Android Studio。如果还没有用 Android Studio 打开过项目，会看到欢迎页，单击 New Project。如果已经用 Android Studio 打开过项目，那么可单击菜单中的 File→New Project 命令来创建一个新的项目。

在弹出的窗口（Configure your new project）中填入内容，如图 2-1 所示，下面简单解释所填各项的含义：

- Application name 是想呈现给用户的应用名称。
- Company Domain 是包名限定符，通常填写公司域名，Android Studio 会将这个限定符应用于每个新建的 Android 项目。
- Package name 是应用的包命名空间（同 Java 的包的概念），该包名在同一 Android 系统上所有已安装的应用中具有唯一性，用户可以独立地编辑该包名。
- Project location 是操作系统存放项目的目录。

图 2-1　新建项目

单击 Next 按钮，在弹出的 Select the form factors your app will run on 窗口选中 Phone and Tablet 复选框，如图 2-2 所示。其中，Minimum SDK 处采用默认值 API 14: Android 4.0 （IceCreamSandwich），Minimum SDK 表示应用支持的最低 Android 版本，为了支持尽可能多的设备，应该设置为能支持应用核心功能的最低 API 版本。如果某些非核心功能仅在较高版本的 API 支持，则可以只在支持这些功能的版本上开启它们。

单击 Next 按钮，在接下来弹出的窗口中选择 Empty Activity，单击 Next 按钮，在弹出的 Customize the Activity 窗口填写 Activity Name 和 Layout Name，此处采用默认命名，即 Activity Name 为 MainActivity，对应的 Layout Name 为 activity_main，



单击 Finish 按钮完成创建。

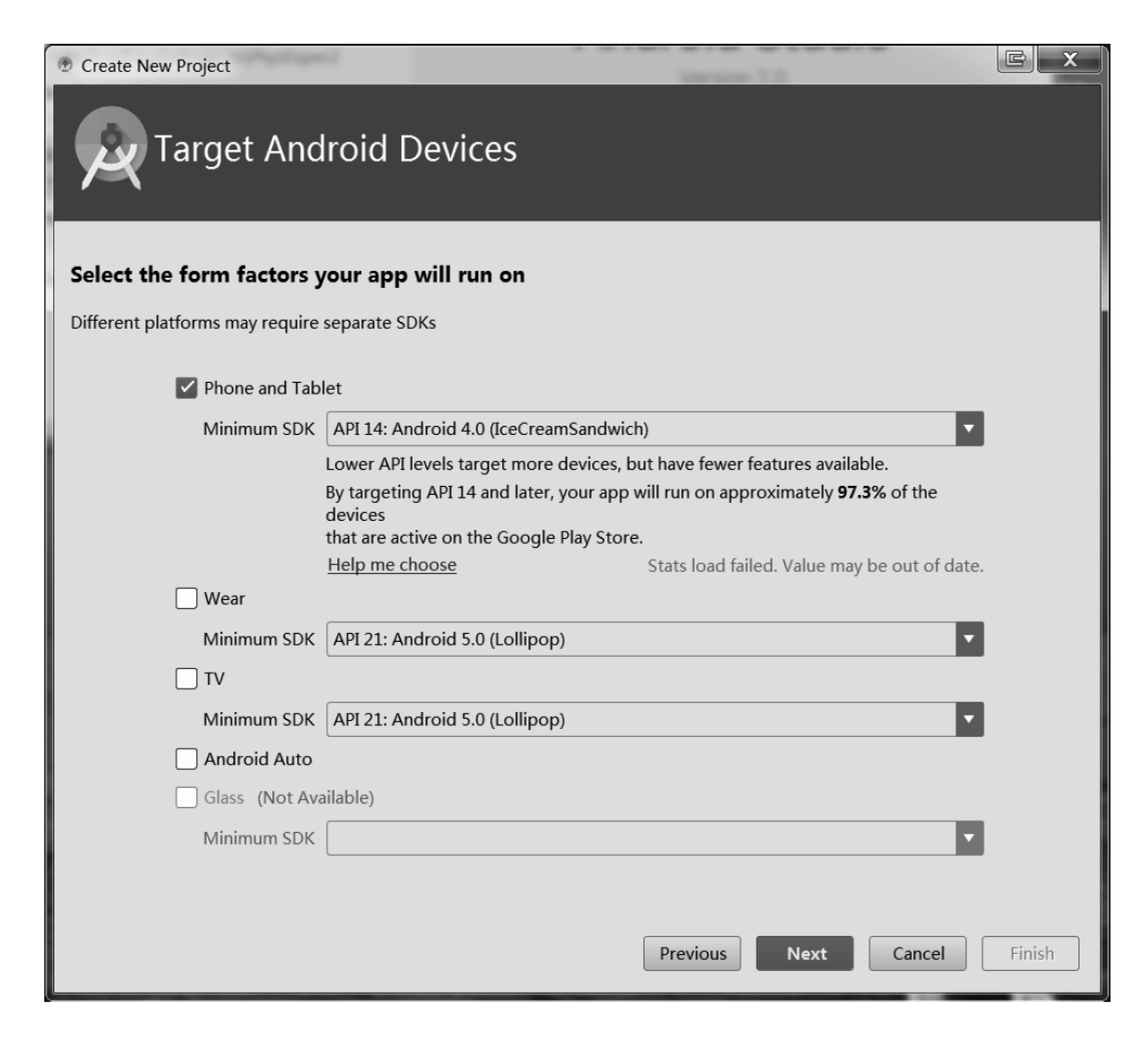


图 2-2　配置目标设备信息

至此，你已经成功创建了一个基础的 Hello World 项目，接下来将从这个项目入手，介绍 Android 文件目录结构中最重要的部分。

2.2　Android 目录结构

在 Android Studio 左侧的 Project 工具窗口可呈现当前项目中的所有包、目录和文件。在左上角选择 Android 模式，窗口会隐藏暂时不用的 build 等文件夹，只显示 app 文件夹和一些重要的 gradle 文件，如图 2-3 所示。下面通过该图对 Android 项目结构的基本内容进行介绍。

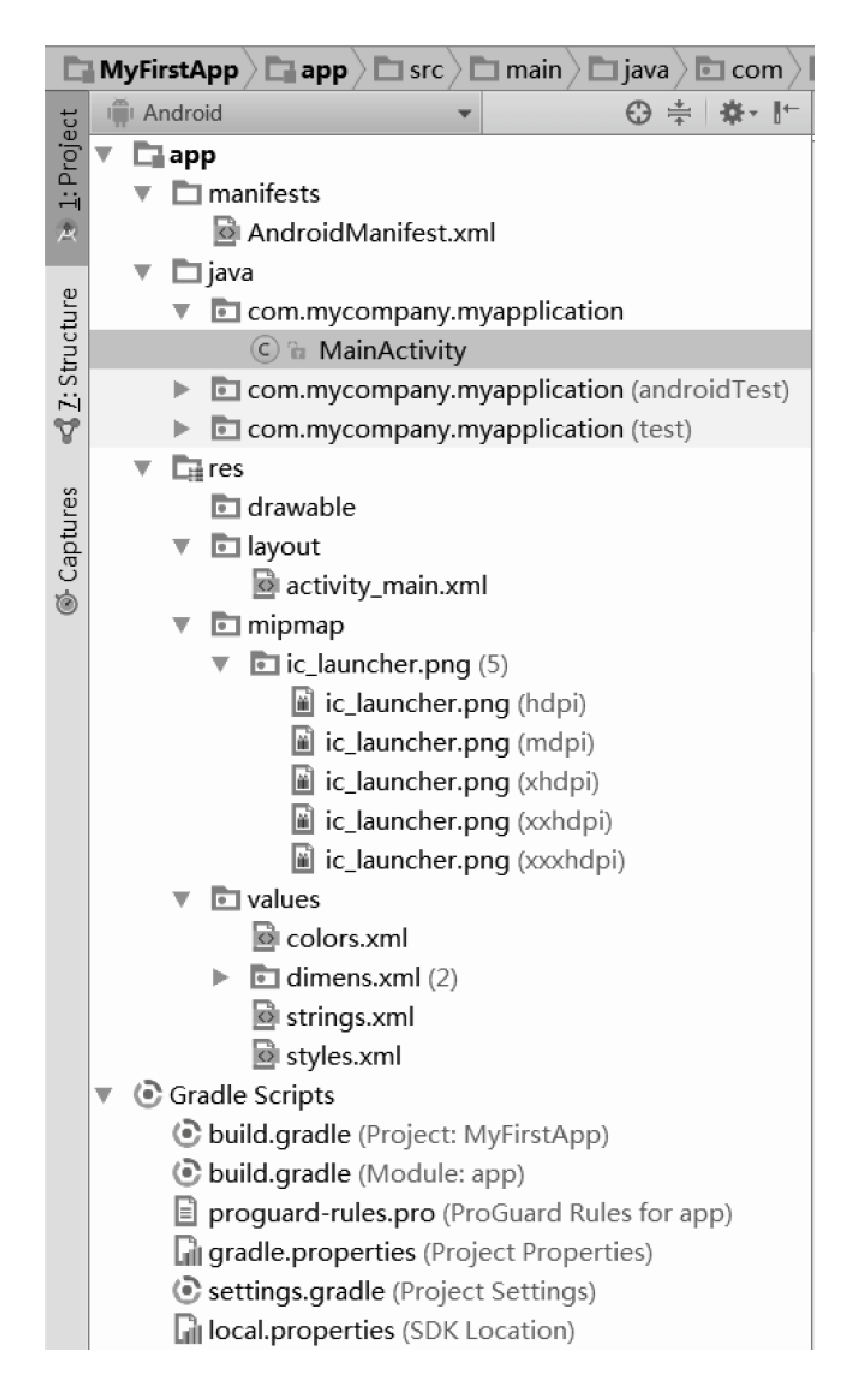


图 2-3　Android 目录结构

1. java

毫无疑问，java 目录是放置所有 Java 代码的地方，展开第一个包之后将看到刚刚通过填写名称创建的 MainActivity 文件就在里面。

2. res

可以看到，这个目录下的内容非常多，在项目中使用到的所有图片、布局、字符串等资源都要存放在这个目录下。当然这个目录下还有很多的子目录，图片放在 drawable 和 mipmap 目录下，其中 mipmap 能够对图片缩放提供一定的性能优化，布局放在 layout 目录下，字符串、颜色等资源放在 values 目录下。

3．manifests

这个目录下有整个 Android 项目的配置文件 AndroidManifest.xml，程序中定义的所有四大组件都需要在这个文件里注册。另外还可以在这个文件中给应用程序添加权限声明，也可以重新指定创建项目时指定的程序最低兼容版本和目标版本。由于这个文件以后会经常用到，在此仅对其文件结构做简要介绍，之后用到的时候再对其做进一步的详细说明。

如图 2-4 所示，AndroidManifest.xml 由一个根 manifest 标签构成，该标签带有一个设置项目包的 package 属性，通常还包含一个 xmlns:android 属性来提供文件内使用的某些系统属性。manifest 标签包含了一些节点，下面列举常用的节点标签。



图 2-4　AndroidManifest.xml 文件

application：一个清单只能包含一个 application 节点，它使用各种属性来指定应用程序的各种元数据（包括标题、图标和主题），同时还作为一个可包含活动、服务、内容提供器和广播接收器标签的容器，用来指定应用程序组件。

activity：应用程序显示的每一个 Activity 都要求有一个 activity 标签，并使用

android:name 属性来指定类的名称。这必须包含核心的启动 Activity 和其他所有可以显示的屏幕或者对话框。启动一个没有在清单中定义的 Activity 时会抛出运行时异常。每一个 Activity 节点都允许使用 intent-filter 子标签来指定哪个 Intent 启动该活动（Intent 将在本书第 6 章详细介绍）。

service：应用程序中使用的每一个 Service 都要有一个 service 标签，它同样也支持使用 intent-filter 子标签来支持后面的运行时绑定。

provider：provider 标签用来说明应用程序中的每一个内容提供器，内容提供器将在本书第 8 章进行详细介绍。

receiver：通过添加 receiver 标签，可以注册一个广播接收器（Broadcast Receiver）而不用事先启动应用程序。广播接收器就像全局事件监听器，一旦注册之后，无论何时，只要与它相匹配的 Intent 被应用程序广播出来，它就会立即执行。通过在声明中注册一个广播接收器，可以使这个进程实现完全自动化。如果一个匹配的 Intent 被广播了，则应用程序就会自动启动，并且你注册的广播接收器也会开始运行。关于广播接收器的具体内容将在本书第 6 章详细介绍。

uses-permission：作为安全模型的一部分，uses-permission 标签声明了那些由你定义的权限，而这些权限是应用程序正常执行所必需的。在安装程序的时候，你设定的所有权限将会告诉给用户，由他们来决定同意与否。

permission：在清单中可以使用 permission 标签来创建访问某个应用程序组件的权限定义，然后应用程序组件就可以通过添加 android:permission 属性来要求这些权限。此后，其他的应用程序则必须在它们的清单中包含 uses-permission 标签，并且通过授权之后才能使用这些受保护的组件。

instrumentation：instrumentation 标签提供一个框架，用来在应用程序运行时在活动或者服务上运行测试。它们提供了一些方法来监控应用程序及其与系统资源的交互，因此，对于应用程序所创建的每一个测试类，都需要创建一个新的 instrumentation 节点。

图 2-5 展示了 AndroidManifest 文件各节点的层次结构。

上述是整个项目的目录结构中最关键的几部分，当然，这里的介绍十分粗略，甚至还有项目结构的许多部分都未涉及，以至于你可能仍有许多疑问。不要担心，当你完成了整个 Android 知识体系的学习，再回来看目录结构的这些部分，就会觉得十分清晰和简单。

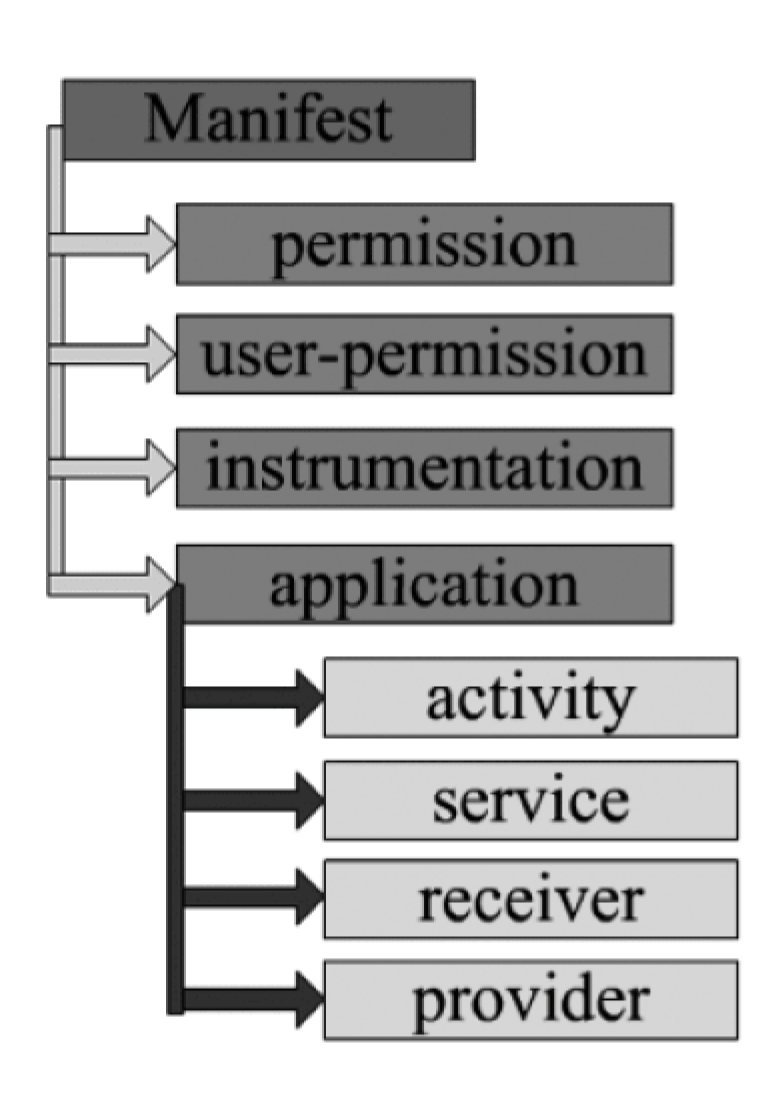


图 2-5 Android Manifest 结构

2.3 Android 项目资源

本节详细介绍 res 目录中的资源，Android 将非代码资源和代码分离开来，从简单的字符串、颜色这样的值到更复杂的资源（如图片、动画、主题和菜单乃至布局），从而使它们变得更易于维护、更新和管理，使我们可以通过轻松地定义多种可选的资源值来支持国际化需求，以及包含不同的资源来支持硬件的变化，特别是屏幕尺寸和分辨率的变化。

2.3.1 创建资源

应用程序的资源根据资源类型存放在 res 目录的不同子文件下，当编译应用程序时，这些资源会被尽可能高效地编译和压缩，并包含到应用程序包中。这个过程还会创建一个 R 类文件，它包含了对加入到项目中的每一个资源的引用，因此可以在代码中方便地引用资源。在任何情况下，资源的文件名都应该只包含小写字母、数字、点（.）和下画线（_）。

注意：drawable 目录和 mipmap 目录按照屏幕分辨率可分为 hdpi（高分辨率）、mdpi（中分辨率）、xhdpi（超高分辨率）、xxhdpi（超超高分辨率）、xxxhdpi（超超超高分辨率）。创建资源时通常将相同的图片按照不同的分辨率制作多份分别存放在上述对应的目录下，即从原始的矢量图像资源着手，然后根据如表 2-1 所示的尺寸比例，生成各种密度下的图像。

表 2-1　分辨率对应尺寸比例

分　辨　率	尺 寸 比 例
xhdpi	2.0
hdpi	1.5
mdpi	1.0（基准）
ldpi	0.75

这意味着，如果针对 xhdpi 的设备生成了一张 200×200 的图像，那么应该为 hdpi 生成 150×150，为 mdpi 生成 100×100 的图像，分别放入对应的目录下，以便在引用资源时系统根据屏幕的分辨率选择恰当的资源。

2.3.2　使用资源

除了自己创建的资源外，Android 平台还提供了多种系统资源供开发者在应用程序中使用。既可以在程序代码中直接使用这些资源，也可以在其他资源中引用这些资源。

1．在代码中使用资源

在代码中使用静态 R 类来访问资源，R 类是在编译项目时基于资源自动生成的类，可以在工程文件夹下的/app/build/generated/source/r/debug/com/mycompany/myapplication 中找到它。

对于已为其至少定义了一个资源的资源类型，R 类中会包含一个对应的静态子类，例如 R.string 和 R.drawable。R 中的每一个子类都将相关的资源表示为变量形式，变量的名字即资源的标识符，变量的值是一个整数，对应每个资源在资源表中的位置，而非资源本身的实例。

当一个构造函数或方法（例如下面的 setContentView）接受一个资源标识符的参数时，可以传入对应的资源变量：

```
// Inflate a layout resource.
setContentView(R.layout.main);
```

当需要一个资源本身的实例时，还需要使用特定的方法把它们从资源表中提取出来。资源表为 Resource 类的一个实例。首先，可以通过在应用程序的上下文中使用 getResources 方法来获取 Resource 实例：

Resources 类为每一个可用的资源包含了 get 方法，之所以要获取 Resources 类的实例，是因为这些方法要在应用程序的当前资源表中进行查找，所以它们不能是静态的，下面的代码展示了如何获取各类资源的实例：

```
Resources myResources = getResources();
CharSequence styledText =
myResources.getText(R.string.stop message);
Drawable icon =
myResources.getDrawable(R.drawable.app icon);
int opaqueBlue =
myResources.getColor(R.color.opaque blue);
float borderWidth =
String[] stringArray;
stringArray =
myResources.getStringArray(R.array.string_array);
```

2．在资源内引用资源

也可以使用资源的引用作为其他 XML 资源的属性值，这常常用在 layout（布局）和 style（样式）中，如下面的代码所示，使用@[packagename:]resourceType/resourceIdentifier 就可以在一个资源中引用另一个资源。默认情况下，Android 认为正在使用的是同一个包中的资源，如果使用的是其他包中的资源，那么需要完全限定包的名称。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout width="match parent"
android:layout height="match parent"
android:padding="@dimen/standard border">
    <EditText
     android:id="@+id/myEditText"
     android:layout width="match parent"
     android:layout_height="wrap_content"
     android:text="@string/stop_message"
     android:textColor="@color/opaque_blue"/>
</LinearLayout>
```

Android 的框架也提供了许多本地资源，在代码中使用系统资源与使用自己创建的资源方法类似，不同的是需要使用 android.R 类而不是应用程序特定的 R 类；

在 XML 中访问系统资源同样与引用自己创建的资源方法类似，不同的是需要限定 android 作为包的名称。

2.3.3 资源本地化

全世界的 Android 设备有着各种各样的大小和尺寸，为了能够在各种 Android 平台上使用，我们的 app 需要兼容各种不同的设备类型。Android 的资源树包含着对应于各种可选的硬件配置、语言和位置的值。当应用程序启动时，Android 可以自动从资源树中选择正确的资源值，从而实现诸如根据用户的语言和国家定制文本、根据屏幕密度改变图片、根据屏幕的尺寸和方向改变布局等功能。

1. 适配不同的语言

为支持多国语言，需要在 res/中创建一个额外的 values 目录以连字符和 ISO 国家代码结尾命名，比如 values-es 是为语言代码为 es 的区域设置的简单的资源文件的目录，并添加不同区域语言的字符串值到相应的字符串资源文件。

```
res/
      values/
          strings.xml
      values-es/
          strings.xml
      values-fr/
          strings.xml
```

Android 会在运行时根据设备的区域设置，加载相应的资源。例如下面列举了几种不同语言对应的资源子目录和字符串资源文件。

（1）对应英语（默认语言）的资源子目录和字符串资源文件/values/strings.xml 如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">My Application</string>
    <string name="hello world">Hello World!</string>
</resources>
```

（2）对应西班牙语的资源子目录和字符串资源文件/values-es/strings.xml 如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">Mi Application</string>
<string name="hello world">Hola Mundo!</string>
</resources>
```

（3）对应法语的资源子目录和字符串资源文件/values-fr/strings.xml 如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">Mon Application</string>
<string name="hello world">Bonjour le monde !
</string>
</resources>
```

2．适配不同的屏幕

为了针对不同的屏幕去优化用户体验，需要为每一种将要支持的屏幕尺寸创建唯一的 XML 文件。每一种 layout 需要保存在相应的资源目录中，目录以-<screen_size>为后缀命名。例如，对大尺寸屏幕，一个唯一的 layout 文件应该保存在 res/layout-large/中。例如下面的工程包含一个默认 layout 和一个适配大屏幕的 layout。

```
res/
    layout/
        main.xml
layout-large/
        main.xml
```

layout 文件的名字必须完全一样，为了对相应的屏幕尺寸提供最优的 UI，文件的内容有所不同。像之前介绍的那样简单引用，系统会根据 app 所运行的设备屏幕尺寸，在与之对应的 layout 目录中加载 layout。

2.4　Gradle 详解

Android Studio 使用 Gradle 构建系统，这是一种从源文件集合中构建软件包的通用工具。对于工程的每个模块以及整个工程，都有一个 build.gradle 文件，通

常你只需要关注模块的 build.gradle 文件，该文件包含 compiledSdkVersion、defaultConfig、buildTypes、dependencies 几个重要部分。

其中，compiledSdkVersion 是将要编译的目标 Android 版本，此处默认为你的 SDK 已安装的最新 Android 版本，仍然可以使用较老的版本编译项目。但是把该值设为最新版本，可以使用 Android 的最新特性并在最新的设备上优化应用来提高用户体验。

接下来的 defaultConfig{ … }闭包中通常包括以下几部分：

- applicationId——创建新项目时指定的包名。
- minSdkVersion——创建项目时指定的最低 SDK 版本(使用 API level 表示)。
- targetSdkVersion——测试过的应用支持的最高 Android 版本（同样用 API level 表示）。
- versionCode——管理控制的版本号码，必须使用整数值而非字符串。
- versionName——对外发布的版本名称，注意与上述版本号不同，值为字符串。

buildTypes{ … }闭包控制构建系统的输出，使用这个块可以定义用于发布到 GooglePlay 商店的特殊配置。

dependencies{ … }闭包简化了 Android 中的项目依赖，当项目中的代码需要调用另一个 Gradle 项目或 Android Studio 模块中的代码时，只需要在主项目中声明依赖即可将代码绑定在一起。因此需要了解依赖的声明方法，如下面的代码所示：

```
1  dependencies {
2  compile fileTree(dir: 'lib', include: ['*.jar'])
3  compile 'com.android.support:support-v4:20.+'
4  }
```

第一行的 compile 告知 Gradle 在编译过程中获取 libs 文件夹下的所有.jar 文件，第二行的 compile 告知 Gradle 找到版本 20 或者更高的 support-v4 库并让它在项目中可用。Gradle 会根据需要从 Internet 下载依赖模块，将其加入到编译器的 classpath 中，并将它们打包到最后的 App 中。

2.5 项目调试与运行

2.5.1 Android 项目运行

2.1 节创建了一个 Android 的 Hello World 项目，该项目默认包含一系列源文

件，可以立即运行应用程序。

如何运行 Android 应用取决于两件事情：是否有一个 Android 设备和是否正在使用 Android Studio 开发程序。本节将介绍如何使用 Android Studio 在真实的 Android 设备或者 Android 模拟器上安装并且运行应用。

如果有一个真实的 Android 设备，就可以容易地在自己的设备上安装和运行应用程序：

首先把手机设备用 USB 线连接到计算机上。如果是在 Windows 系统上进行开发的，可能还需要安装设备对应的 USB 驱动，开启设备上的 USB 调试选项，在大部分运行 Android 系统的设备上，这个选项位于“设置”→“开发人员选项”里。从 Android 4.2 开始，开发人员选项在默认情况下是隐藏的，如果想让它可见，需要去“设置”→“关于手机”单击版本号七次，返回后即可找到开发人员选项。在 Android 4.0 或更新版本中，这个选项在“设置”→“开发人员选项”中。

在 Android Studio 选择要运行的项目，单击工具栏里的 Run 按钮。当出现 Choose Device 窗口时，选择 Choose a running device 单选框，单击 OK 按钮。Android Studio 会把应用程序安装到我们的设备中并启动应用程序，项目运行成功，如图 2-6 所示。

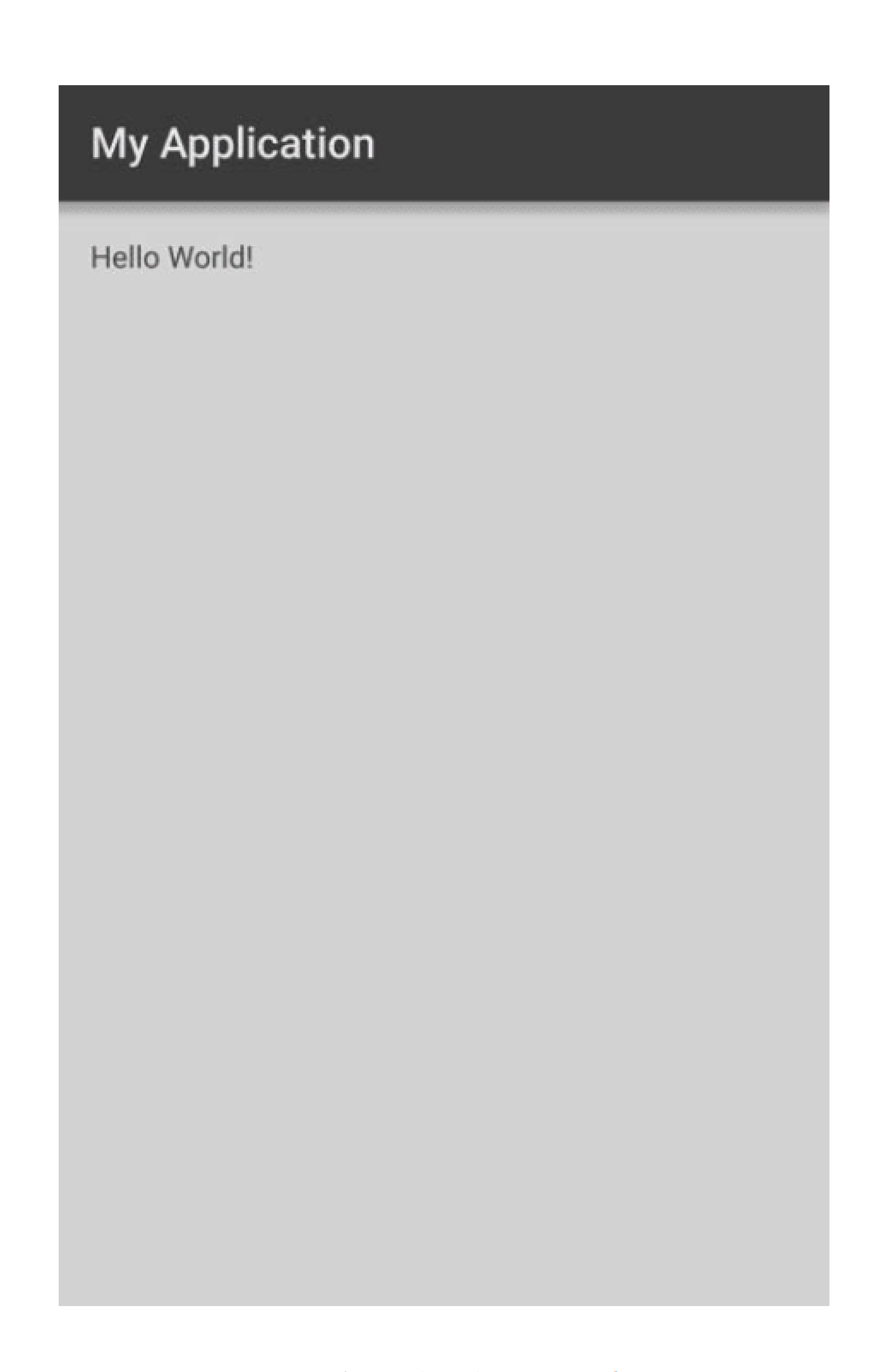


图 2-6　默认的应用程序界面

如果需要在模拟器上运行项目，首先要创建一个 Android Virtual Device（AVD）。使用 Android Studio，单击 Tools→Android→AVD Manager 选项，在弹出的 AVD Manager 面板中，单击 Create Virtual Device，在 Select Hardware 窗口中，选择一个设备（如 Nexus 6），单击 Next 按钮，然后选择列出的合适系统镜像，校验模拟器配置，单击 Finish 按钮。

在 Android Studio 中选择要运行的项目，单击工具栏里的 Run 按钮。当 Choose Device 窗口出现时，选择 Launch emulator 单选框，从 Android virtual device 下拉列表框中选择创建好的模拟器，单击 OK 按钮。模拟器启动需要几分钟的时间，启动完成后，解锁即可看到程序已经运行到模拟器屏幕上了。

2.5.2 Android 项目调试

App 越复杂，出现错误的可能性就越大，App 崩溃、在某些情况下无法运行或者执行的任务与期望不符都会使开发者感到很沮丧。因此，我们有必要在开始时简单介绍一下 Android 中日志工具的使用方法，这对以后的 Android 开发调试之旅会有极大的帮助。

日志在任何项目的开发过程中都会起到非常重要的作用，在 Android 项目中，如果想要查看日志，就必须使用 LogCat 工具。Android App 在一台机器上开发但在另一台机器上运行，这使得它的日志系统与其他平台上的稍有不同，打印的输出并不在代码运行的设备上显示，而是保存在一系列循环缓冲区中。这些缓冲区包括 radio（与音频和电话相关的消息）、events（系统事件消息，例如服务的创建和销毁的通知）、main（主日志输出）。从所有这些事件中查看日志无异于大海捞针，因此需要学习如何使用各种操作和标识来减少输出量。

日志中的每条消息都有一个标签，标签是一个短字符串，通常用来标识消息发出的组件，每条消息还包含一个相关的优先级：V（Verbose 最低优先级）、D（Debug）、I（Info）、W（Warning）、E（Error）、F（Fatal）。Android 中的日志工具类 Log（android.util.Log）提供了如下几个方法来供开发者打印日志：

（1）Log.v()——这个方法用于打印那些最为琐碎的、意义最小的日志信息。对应级别 verbose。

（2）Log.d()——这个方法对应级别 Debug，用于打印一些对调试和分析问题有帮助的调试信息。

（3）Log.i()——这个方法用于打印一些比较重要的数据，这些数据应该是开

发者非常想看到的，可以帮助分析用户行为的数据。

（4）Log.w()——这个方法用于打印一些警告信息，提示程序在这个地方可能会有潜在的风险,最好修复一下这些出现警告的地方。

（5）Log.e()——这个方法对应级别 error，用于打印程序中的错误信息。

以 2.1 节创建的程序为例，打开 MainActivity，在 onCreate()方法中添加一行打印日志的语句：

```
protected void onCreate(Bundle savedInstanceState) {
Log.d(this.getClass().getSimpleName(),"onCreate()");
super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
```

Log.d 方法接收两个参数：第一个参数是 tag，主要用于对打印信息进行过滤，此处使用类名作为标签；第二个参数是 message，即想要打印的消息字符串，此处为方法名 onCreate()。

现在，重新运行一下这个项目，单击 Android Studio 底部的 6 号选项卡，使用内置的设备 LogCat 查看器。在此视图右上角，可以看到三个重要的过滤器控件：最左侧的 Log level 下拉列表根据优先级进行过滤操作，此处设置为 Debug 则可显示包含 Debug 或者更高优先级的所有信息；相邻的文本输入控件可以将消息限定为包含所输入文本的内容；右侧的下拉列表则包含了一系列预置的过滤器和开发者的自定义过滤器。在输出中，不仅可以看到打印日志的内容和标签名，还包括程序的包名、打印的时间以及应用程序的进程号。如果 LogCat 中并没有打印出任何信息，可能是因为当前的设备失去焦点了，这时只需要进入到 DDMS 视图，在 Devices 窗口中单击一下当前的设备即可。最终效果如图 2-7 所示。



图 2-7　LogCat

习　题　2

1．请尝试新建一个带空白 Activity 的项目，并将“Hello World!”字符串改为“这是我的第一个安卓应用程序”。

2．请阐述 res 目录下的 drawable 和 mipmap 目录的异同。

3．请尝试新建一个带空白 Activity 的项目，并使得其在英文环境下显示“Hello World!”，在中文环境下显示“你好，世界！”。

4．Android 有很多第三方的开源包，请尝试使用 Gradle 添加到你的项目中。

5．请为第 3 题生成项目的 MainActivity 再新建一个布局，使得其在高分辨率设备上使用新布局，在低分辨率设备上使用旧布局。

第 3 章　初级 UI

3.1　Android UI 基本概念

用户界面（User Interface，UI）是用户与设备之间进行信息交流的直接媒介，是决定用户体验最重要的部分。相比于早期的计算机的主要交互界面——批处理界面和命令行界面，现在更为流行的是更简单直接的用户图形界面（Graphical User Interface, GUI）。GUI 简单易用，受众面广，直接推动了个人计算机的发展。目前，主流大众的操作系统都采用了 GUI，安卓也不例外。

通常 GUI 上会放置各种组件，这些组件通过巧妙的设计，便能组成灵活美观的界面。Android 程序的 UI 组件分为 widget 控件和 layout 组件两大类，这两类的根类都是 View 类。

- widget 控件：UI 的最基本单位，即不能在这类组件中放入其他 UI 组件。常见的 widget 组件有 Button（按钮）、TextView（文本标签）、EditText（文字输入框）等。
- layout 组件：布局组件，像容器一般，其中可以加入其他 layout 组件或 widget 组件。常用的 layout 组件有 LinearLayout（线性布局）、RelativeLayout（相对布局）、FrameLayout（框架布局）、TableLayout（表格布局）、GridLayout（网格布局）等。

View 类的常用 xml 文件元素属性如表 3-1 所示。

表 3-1　View 类属性

属　　性	对 应 方 法	说　　明
android.id	setId(int id)	设置组件的标识符
android.background	setBackGround(int color)	设置背景颜色
android.visibility	setVisibility(int)	设置组件的可见性
android.clickable	setClickable(Boolean)	设置组件是否响应单击事件
android.alpha	setAlpha(float)	设置组件的透明度

续表

属　　性	对 应 方 法	说　　明
android.layout_weight	setHeight(int)	设置组件的宽度，一般有 match_parent 和 wrap_content 两个选项
android.layout_height	setWidth(int)	设置组件的宽度，一般有 match_parent 和 wrap_content 两个选项
	findViewById(int id)	与 id 对应的组件匹配

基于 MVC（Model-View-Controller）模型，Android 程序开发采用界面设计与程序逻辑分离的策略。开发者应该使用 XML 文件对用户界面进行描述，将资源文件独立保存在资源文件夹中。Android 的用户界面描述非常灵活，允许定义用户界面组件的大小、位置、外观甚至触发事件。

> **更上一层**　MVC 模型包括处理用户输入的控制器（Controller）、显示图像的视图（View）和模型（Model）。模型是应用程序的核心，数据和代码都保存在模型中。

Android 的用户界面框架采用单线程用户界面（Single-threaded UI）的模式。在这种模式下，控制器从事件队列中获取事件和视图在屏幕上绘制用户界面采用的是同一进程。因此，用户不需要在控制器和视图之间进行同步，而且所有事件的处理都是按照其加入事件队列的顺序进行的，也就是事件处理函数具有原子性。但此模式也有弊端。子线程中不允许直接修改用户界面。而且如果事件处理函数过于复杂，可能会使用户界面失去响应，因此复杂的事件处理工作应该交给后台线程处理。

3.2　基 本 控 件

3.2.1　TextView

TextView（文本标签）用于显示文本字符，是最常用的 UI 组件之一，支持多行文本和自动换行。常用的方法如下：

- getText()——获取文本标签的文本内容；
- setText(CharSequence text)——设置文本标签的文本内容；
- setTextSize(float size)——设置文本标签的文本大小；
- setText(int color)——设置文本标签的文本颜色。

常用的 XML 文件元素属性如下：

- android:text——文本标签的文本内容；
- android:textSize——文本标签的文本大小；
- android:textColor——文本标签的文本颜色；
- android:typeface——文本标签的字体样式，有 normal、sans、serif、monospace 等选项。

此外，适用于 View 类的元素属性也适用于 TextView 等所有组件，此处不再赘述。

【例 3-1】 演示 TextView 组件的使用方法。

打开 Android Studio，新建一个带空白 Activity 的项目，命名为 TextViewDemo。

打开界面布局文件 activity_main.xml。Android Studio 已经自动在其中生成一个“Hello World!”的 Textview。切换到代码视图，修改 activity_main.xml 的代码如下：

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <RelativeLayout
3    xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      xmlns:tools="http://schemas.android.com/tools"
6      android:layout width="match parent"
7      android:layout height="match parent"
8     …>
9   <TextView
10     Android:id="+id/textView1"
11         android:layout_width="wrap_content"
12         android:layout height="wrap content"
13         android:textSize="20"
14         android:text="string/textViewString" />
15  </RelativeLayout>
```

本书后文代码中的所有省略号均表示由于篇幅所限，部分内容省略不写

该布局文件添加了一个 TextView 组件。第 10 行表明为该组件增设一个 id 为 textView1 的 TextView；第 11、12 行分别指定其宽度和高度；第 13 行设置该文本标签的文本大小；第 14 行指定其文本内容引用 string.xml 下的 textViewString，开发者也可以在布局文件中设置直接字符串，而不引用字符串资源，但这不利于复用和后期修改，不推荐。

在 value 目录下的 string.xml 中添加字符串资源。

```
<resources>
    <string name="app_name">TextViewDemo1</string>
    <string name="textViewString">"TextView 文本标签内容"</string>
</resources>
```

为了在代码中引用控件，需要在代码中引入相应的 android.widget 开发包，然后使用 findViewById(int id)函数通过 id 引用该控件，并将该控件赋值给创建的空间对象。如下编写代码，将 TextView 的文本颜色改为红色。

```
1   package com.example.textviewdemo;
2   import android.graphics.Color;
3   import android.support.v7.app.AppCompatActivity;
4   import android.os.Bundle;
5   import android.widget.TextView;

6   public class MainActivity extends AppCompatActivity {
7       TextView textView;
8       @Override
9       protected void onCreate(Bundle savedInstanceState) {
10          super.onCreate(savedInstanceState);
11          setContentView(R.layout.activity_main);
12          textView=(TextView)findViewById(R.id.textView);
13          textView.setTextColor(Color.RED);
14      }
15  }
```

代码第 7 行声明了一个 TextView 组件对象。第 12 行使用 findViewById(int id)函数关联到对应 id 的组件，但该函数返回的组件类型是 View，需要强制转换为 TextView。第 13 行，通过调用 TextView 的 SetTextColor(int color)函数修改 TextView 的文本颜色为红色。

编译、运行程序，最终界面如图 3-1 所示。

3.2.2　Button 和 ImageButton

Button（按钮）是普通按钮控件，用于处理人机交互事件。用户单击该控件，

能触发相应的响应事件。如果需要在按钮上显示图像，则可以使用 ImageButton（图像按钮）。Button 继承了 TextView 的所有方法和属性，而 ImageButton 继承自 ImageView。设置 ImageButton 的图像时，需要提前将图像文件放置在 res/drawable 目录下，然后在 xml 文件中设置 ImageButton 的 android.src 属性。此外，还可以在 Java 代码中调用 ImageButton 的 setImageSource(int id)方法设置图像。

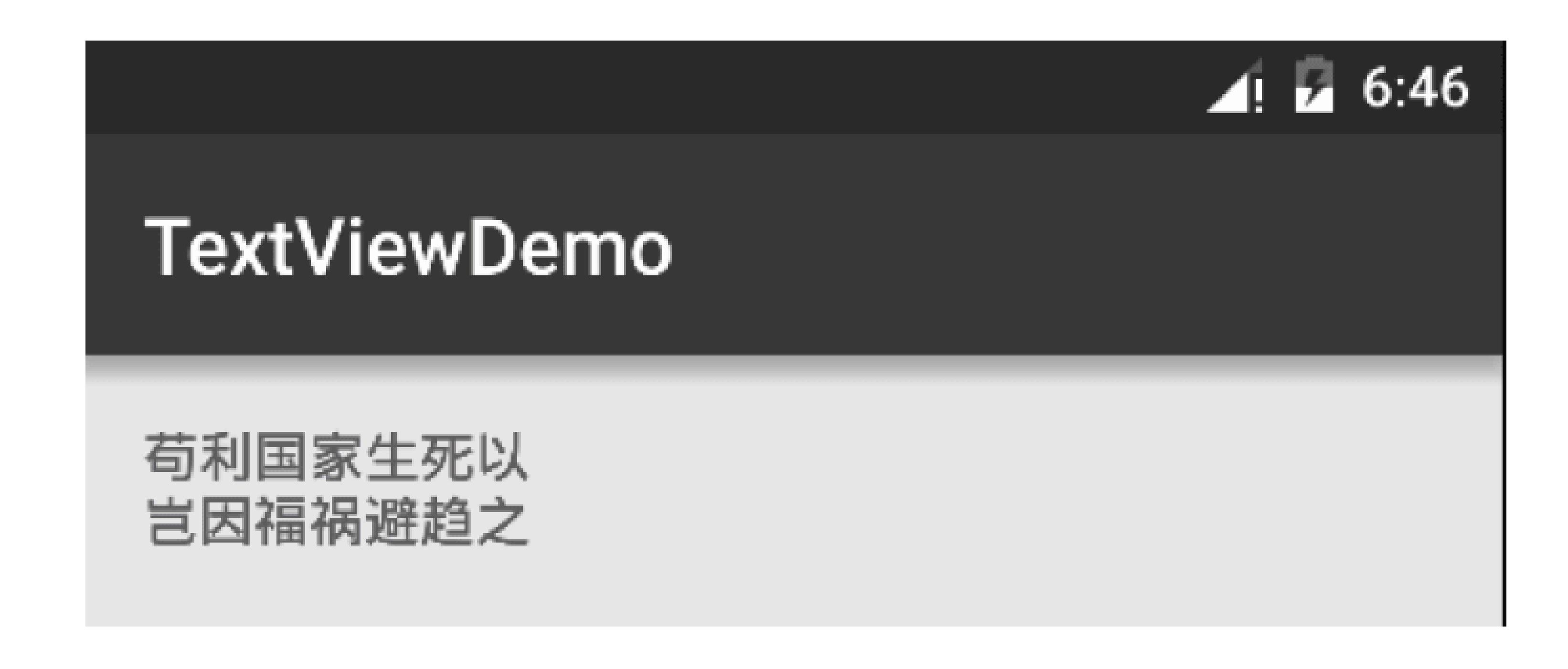


图 3-1　TextView 样式

注册 Button 响应事件的常用方式是实现 OnClickListener 接口。单击按钮时，通过该监听接口触发 OnClick()方法，可以实现特定功能。Button 调用 OnClickListener 接口可用如下方法：

```
按钮对象.setOnClickListener(onClickListener listener)
```

参数 listener 可以传入直接的 OnClickListener 对象，也可以在 Activity 实现 OnClickListener 后，直接传入 this。

【例 3-2】 演示 Button 和 ImageButton 控件的使用方法。

打开 Android Studio，新建一个带空白 Activity 的项目，命名为 ButtonDemo。打开 activity_main.xml，分别添加一个 Button 控件和一个 ImageButton 控件，此外再添加一个 TextView，以便显示按钮单击事件的结果。具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout width="match parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <Button
        android:layout_width="wrap_content"
```

```
9           android:layout_height="wrap_content"
10          android:text="Button"
11          android:id="@+id/button" />
12      <ImageButton
13          android:layout_width="wrap_content"
14          android:layout_height="wrap_content"
15          android:id="@+id/imageButton"
16          android:src="@mipmap/ic_launcher" />
17      <TextView
18          android:layout_width="wrap_content"
19          android:layout_height="wrap_content"
20          android:id="@+id/textView" />
21  </LinearLayout>
```

第 7～11 行声明了一个 ID 为 button 的 Button 控件。第 12～18 行声明了一个 ID 为 ImageButton 的控件，第 16 行表明该 ImageButton 的图像为 mipmap 下的 ic_launcher 图像（此为 Android Studio 默认的应用图标），如果是应用自定义图像，则该语句为 android:src="@drawable/*** "，***为文件名，不带后缀。

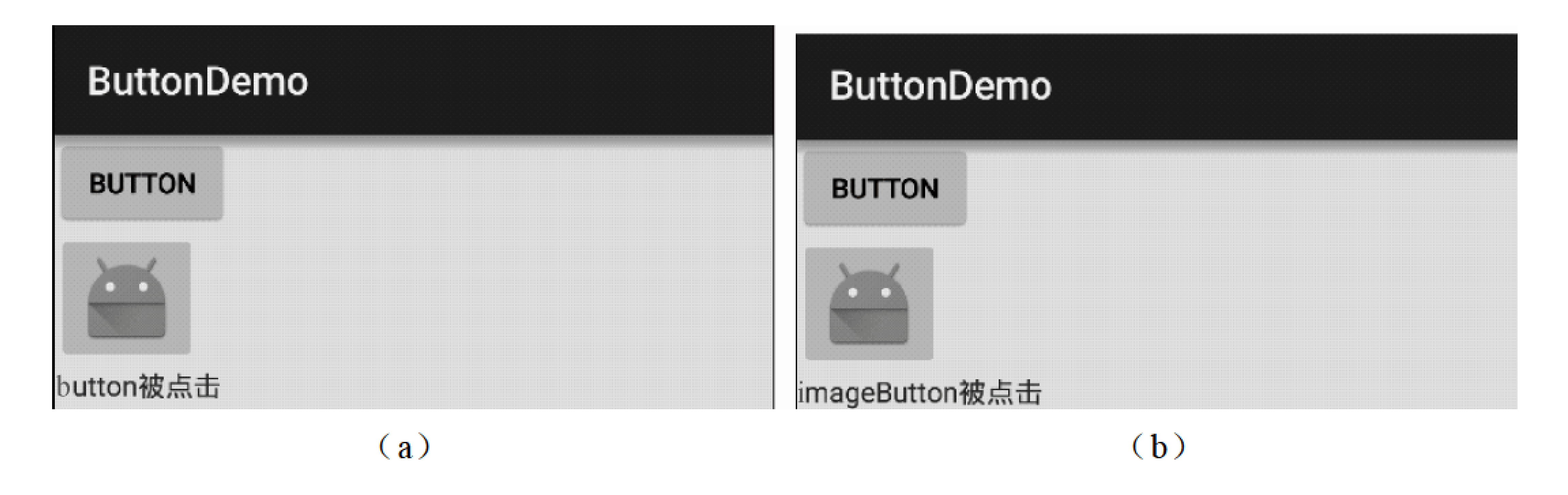
打开 MainActivity.class 文件，引用两个按钮控件，实现并注册单击事件监听器 OnClickListener。修改如下：

```
package com.example.buttondemo;
import …;
public class MainActivity extends AppCompatActivity implements
View. OnClickListener {
    ImageButton imageButton;
    TextView textView;
    Button button;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        imageButton = (ImageButton)findViewById(R.id.imageButton);
        textView = (TextView)findViewById(R.id.textView);
        button = (Button)findViewById(R.id.button);
        imageButton.setOnClickListener(this);
        button.setOnClickListener(this);
```

```
17      }
18      @Override
19      public void onClick(View v) {
20          if(v==imageButton)
21              textView.setText("ImageButton 被点击");
22          else if(v==button)
23              textView.setText("Button 被点击");
24      }
25  }
```

其中第 15、16 行分别注册了 imageButton 和 button 的单击事件监听器。第 19 行到第 23 行实现了接口 View.OnClickListener 的 OnClick()函数，从而实现了该接口。

单击 button 后，运行结果如图 3-2（a）所示，单击 imageButton 后运行结果如图 3-2（b）所示。



（a）　　（b）

图 3-2　Button 和 ImageButton 样式

3.2.3　EditText

EditText（文本编辑框）是用来输入和编辑字符的控件。EditText 继承自 TextView，是一个特殊的具有编辑功能的 TextView 控件，所以 TextView 的方法和 xml 文件元素属性都适用于 EditText，如图 3-3 所示。

EditText 的主要 xml 文件元素属性有：

- android.editable——是否可编辑，取值为 true 或 false。
- android.inputType——输入字符类型，取值主要有 text（文本）、TextPassword（密码）、number（数字）、phone（手机号码）等，可以用“|”连接多个取

值。此外输入类型也可以单独由 android.numeric、android.password、android.phoneNumber 等属性设置。

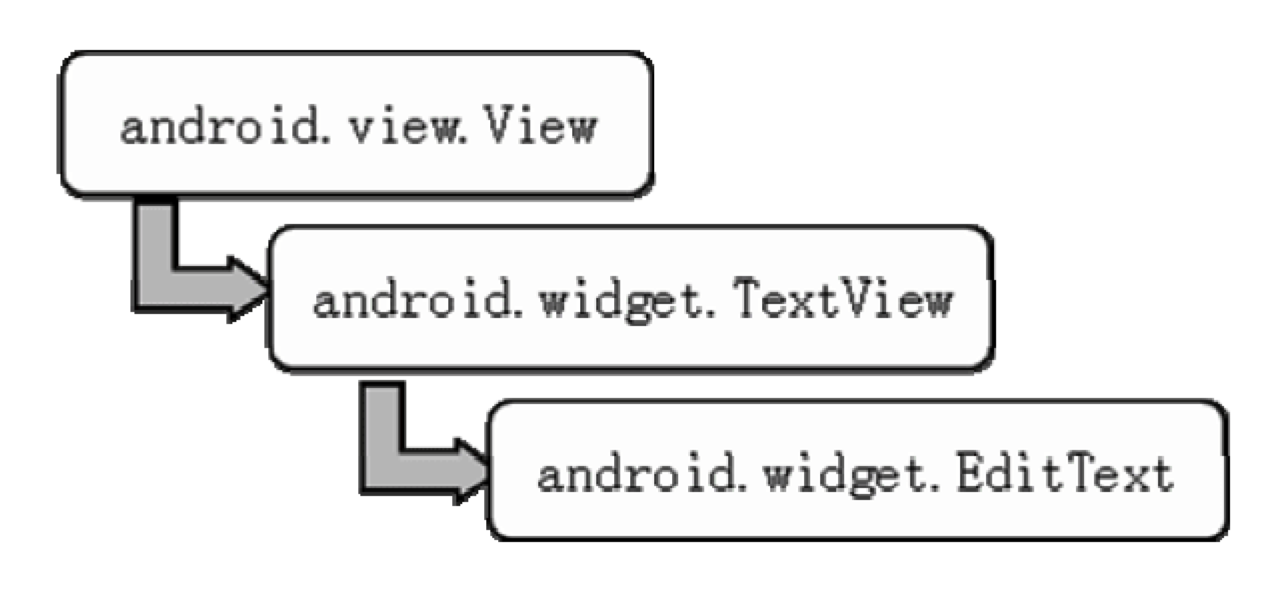


图 3-3　EditText 的继承关系

- android.hint——文本编辑框的提示语。

> **更上一层**　EditText 对象.getText()返回的是 Editable 类型，用于字符串处理时，需调用它的 toString()函数，即 EditText 对象.getText().toString()。

【例 3-3】 演示 EditText 控件的使用方法

打开 Android Studio，新建一个带空白 Activity 的项目，命名为 EditTextDemo。

打开 activity_main.xml，分别添加输入类型为 text（默认）、password 的用户名 EditText 和密码 EditText，再添加一个 Button 作为登录按钮，和一个 TextView 提示登录结果信息。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout height="match parent"
    android:orientation="vertical">
    <EditText
        android:layout_width="match_parent"
        android:layout height="wrap content"
        android:id="@+id/editText1"
        android:hint="用户名"/>
    <EditText
        android:layout width="match parent"
        android:layout height="wrap content"
        android:inputType="textPassword"
```

```
16          android:id="@+id/editText2"
17          android:hint="密码"/>
18      <Button
19          android:layout_width="wrap_content"
20          android:layout_height="wrap_content"
21          android:text="登录"
22          android:id="@+id/button" />
23      <TextView
24          android:layout_width="wrap_content"
25          android:layout_height="wrap_content"
26          android:id="@+id/textView" />
27  </LinearLayout>
```

其中用户名 EditText 的输入类型是默认的 Text（文本），而第 15 行设置密码 EditText 的输入类型是 TextPassword（密码），在输入的时候，只会显示对应个数的*。

修改 MainActivity.java 文件，实现登录按钮的 OnClickListener 接口，设置为当用户名 EditText 输入为 android 且密码 EditText 输入为 123 时，显示登录成功，否则显示登录失败。

```
package com.example.edittextdemo;
import ...;
public class MainActivity extends AppCompatActivity {
    EditText userNameEdit,passwdEdit;
    Button loginButton;
    TextView textView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        userNameEdit = (EditText)findViewById(R.id.editText1);
        passwdEdit  = (EditText)findViewById(R.id.editText2);
        loginButton  = (Button)findViewById(R.id.button);
        textView    = (TextView)findViewById(R.id.textView);
        loginButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
```

```
18  if(userNameEdit.getText().toString().equals("android")&&
19          passwdEdit.getText().toString().equals("123"))
20                      textView.setText("登录成功");
21                  else
22                      textView.setText("登录失败");
23              }
24          });
25      }
26  }
```

此类中实现 Button 的单击事件没有采用 Activity 实现 OnClickListener 接口的方法，而是直接在第 15 行注册监听器传参数时新建一个 OnClickListener 对象，这种方法也是可行的，但在 Button 较多时，代码会显得比较臃肿。

运行结果如图 3-4 所示。

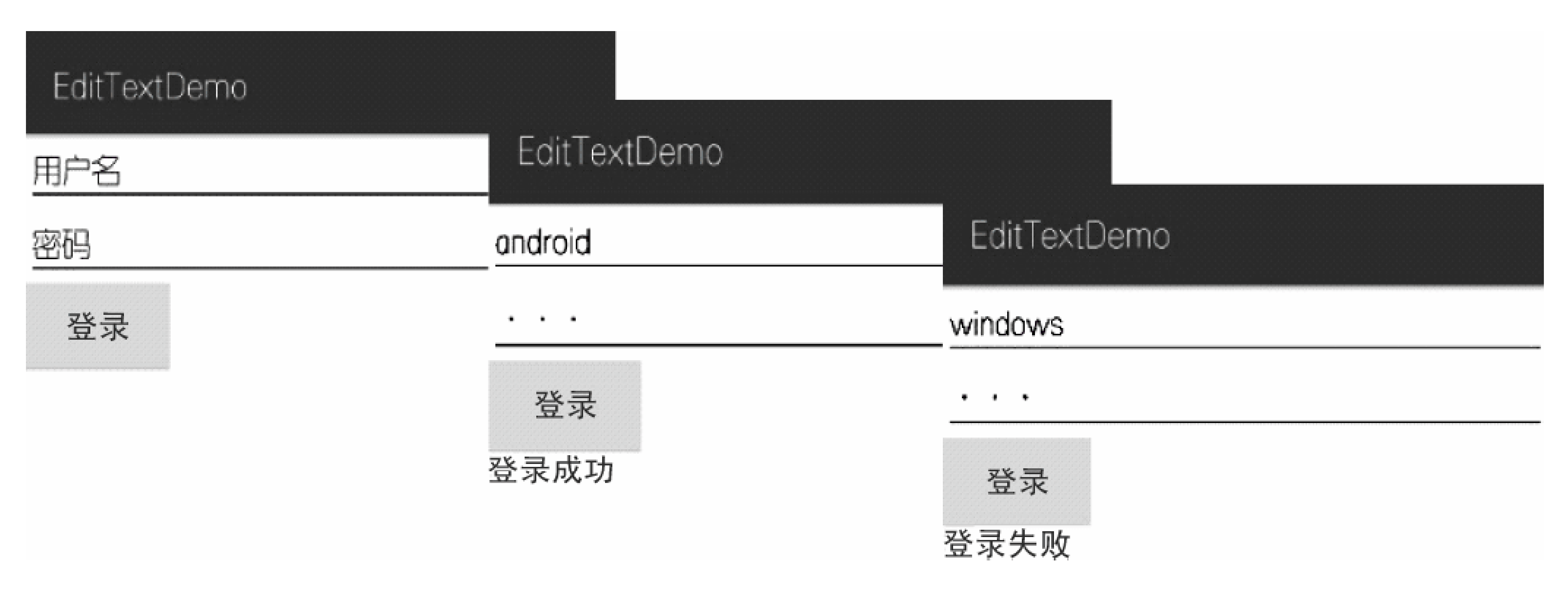


图 3-4　EditText 样式

3.3　Layout 组件

Layout 组件是布局组件，其中可以嵌套容纳其他组件，又称为容器（container）。Layout 组件规定了子组件的摆放规则，因此开发者通过 Layout 组件在 xml 布局文件中控制组件的位置、间距、排列及对齐方式等，从而设计出更具结构化、更为美观，同时兼容多种分辨率的界面。Layout 组件的共同父类是 ViewGroup。

正常情况下，xml 布局文件的根节点是一个 layout 组件，该组件必须有命名空间：xmlns:android="http://schemas.android.com/apk/res/android"。开发者可以根

据需求在该节点下嵌套添加其他组件作为子节点，以丰富界面。

开发者可以设置 Layout 组件的 android:gravity 属性，控制子组件在组件中的位置。或者设置 android:layout_gravity 属性控制组件自身在父组件中的位置。 这两个属性的值可以为 top（上）、bottom（下）、left（左）、right（右）、center_horizontal（水平居中）、center_vertical（垂直居中）。

常用的 layout 组件有 FrameLayout（框架布局）、LinearLayout（线性布局）、RelativeLayout（相对布局）、TableLayout（表格布局）、GridLayout（网格布局）。

3.3.1 FrameLayout

FrameLayout（框架布局）是所有布局中最为简单的一种，也叫帧布局。该布局直接在屏幕上开辟出了一块空白区域，当向其中添加组件的时候，默认情况下，所有的组件都会放置于这块区域的左上角。开发者可以为子组件添加 layout_gravity 属性，从而指定组件的对齐方式。如果有多个子元素，那么后加的子元素就会覆盖先加的子元素。

【例 3-4】 演示 FrameLayout 的用法

打开 Android Studio，新建一个带空白 Activity 的项目，命名为 FrameLayout。按默认方式先后添加两个 TextView，分别命名为“第一层”和“第二层”，再添加名为“第三层”的 TextView，并设置其位置为 FrameLayout 的中央。

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:text="第一层"
        android:textSize="50sp"
        android:background="#e93521"
        android:gravity="center"/>
    <TextView
        android:layout_width="90dp"
        android:layout_height="90dp"
        android:text="第二层"
        android:textSize="30sp"
```

gravity 属性控制 View 中子内容的分布位置

```
17          android:background="#d9e318"
18          android:gravity="center"/>
19      <TextView
20          android:id="@+id/textView"
21          android:layout_width="200dp"
22          android:layout_height="200dp"
23          android:textSize="50sp"
24          android:text="第三层"
25          android:layout_gravity="center"
26          android:background="#18f50d"
27          android:gravity="center"
28          />
29  </FrameLayout>
```

第 5～11 行声明了一个 200dp×200dp 的背景为#e93521（十六进制颜色）、文本为“第一层”的 TextView。第 12～18 行声明了一个 90dp×90dp 的背景为#d9e318、文本为“第二层”的 TextView。后生成的子元素会覆盖先生成的子元素，因此“第二层”会覆盖“第一层”。第 19～28 行生成一个 200dp×200dp 的背景为#18f50d、文本为“第三层”的 TextView，并通过设置其属性 android:layout_gravity 为 center，以达到将其放置在 FrameLayout 中央的效果。最后的运行界面如图 3-5 所示。

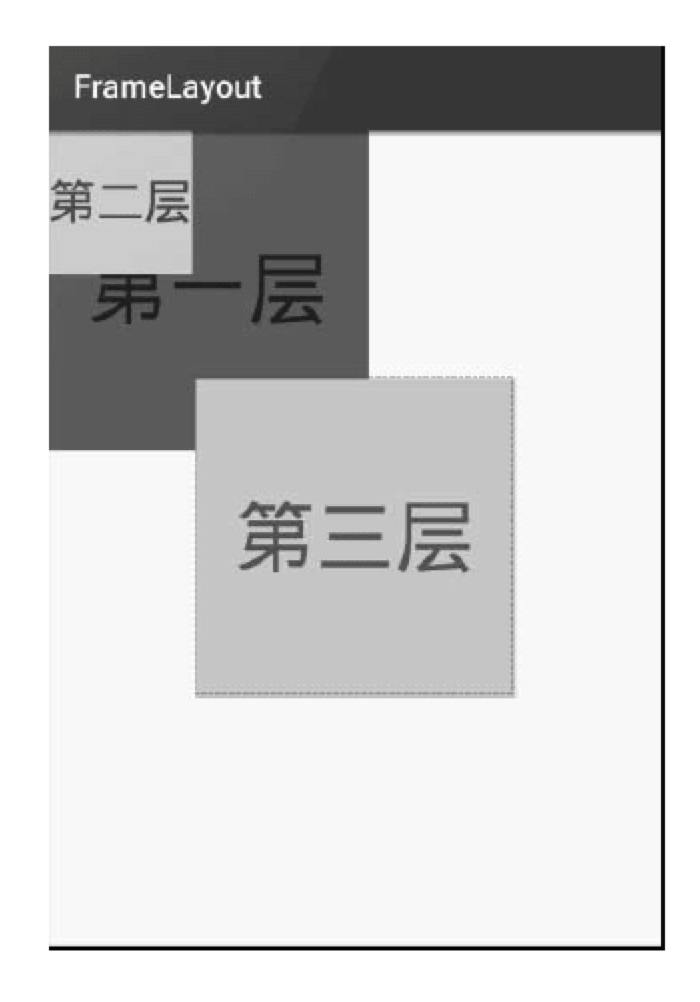
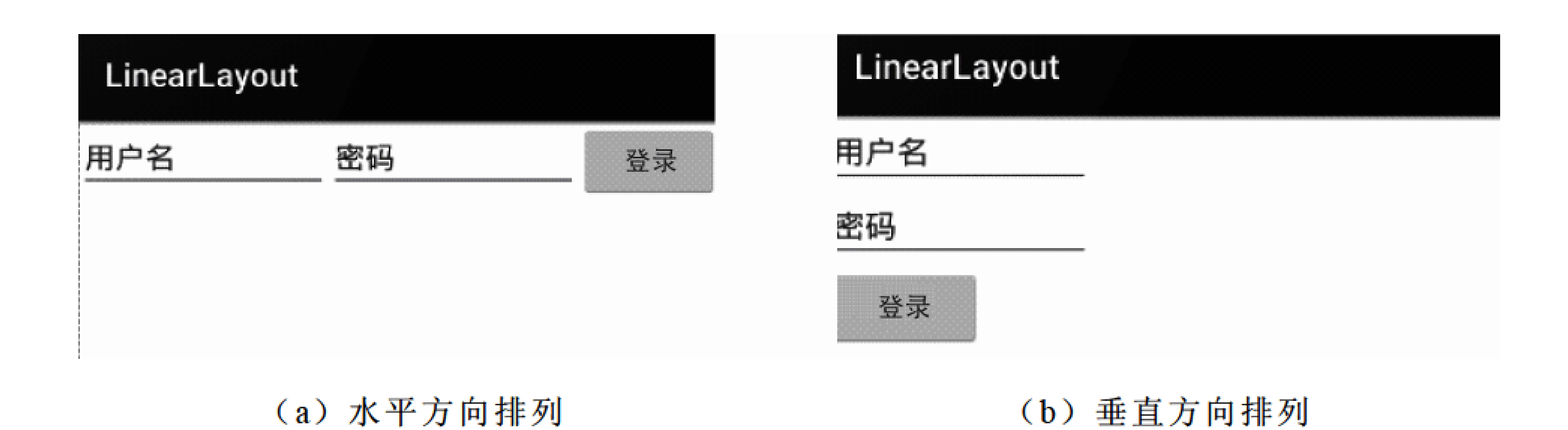


图 3-5　FrameLayout

3.3.2　LinearLayout

LinearLayout（线性布局）是一般开发中最常用的布局组件之一。在

LinearLayout 中，所有的子元素都在水平或垂直方向依次排列。若为垂直方向，则每行只有一个子元素；若为水平方向，则每列只有一个子元素，如图 3-6 所示。默认情况下，LinearLayout 的排列方向为水平。开发者可以设置 android:orientation 属性控制方向，取值可以为 horizontal（水平）或 vertical（垂直）。



（a）水平方向排列　（b）垂直方向排列

图 3-6　LinearLayout 水平和垂直样式

如果开发者想令子组件按比例分配，可以将子组件对应方向的长度设为 0，即水平分布时令 android:layout_width="0dp"，垂直分布时令 android:layout_height="0dp"，然后设置 android.layout_weight 的值。最后组件所占比例为该组件 layout_weight 值除以该组件父组件的所有子元素的 layout_weight 之和。

【例 3-5】 演示如何令 LinearLayout 组件中的子组件按比例分布

打开 Android Studio，新建一个带空白 Activity 的项目，命名为 LinearLayout。打开 activity_main.xml 文件，在根节点 LinearLayout 下分别添加用户名 EditText、密码 EditText 和登录 Button，令其在水平方向按 2:2:1 的比例分布。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">
    <EditText
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:hint="用户名"/>
    <EditText
        android:layout_width="0dp"
        android:layout_height="wrap_content"
```

```
15          android:layout weight="2"
16          android:inputType="textPassword"
17          android:id="@+id/editText2"
18          android:hint="密码"/>
19      <Button
20          android:layout_width="0dp"
21          android:layout height="wrap content"
22          android:layout_weight="1"
23          android:text="登录"
24          android:id="@+id/button" />
25  </LinearLayout>
```

第 6 行设置 LinearLayout 子组件的排列方式为水平。第 10、15、22 行分别设置所在组件的 layout_weight 为 2、2 和 1，所以这三个组件最后所占比例分别为 2/（2+2+1）=40%、40%和 20%。应用程序界面如图 3-7 所示。

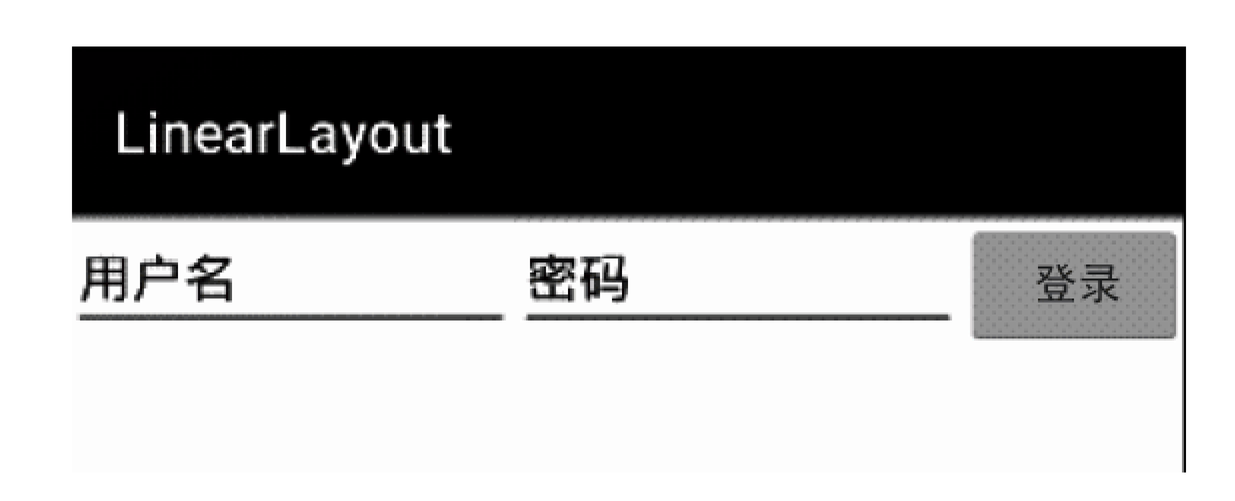


图 3-7　LinearLayout 样式

3.3.3　RelativeLayout

RelativeLayout（相对布局）是一种非常灵活的布局控件，采用相对其他组件的位置的布局方式。RelativeLayout 中通常使用相对父组件的位置或通过 id 关联其他组件，以达到上、下、左、右对齐的目的。

为达到灵活的目的，相对布局有很多属性，如表 3-2 所示。

表 3-2　相对布局的属性

属　　性	取　　值	说　　明
android:centerInParent	true、false	是否在父控件的中央
android:centerInHorizontal	true、false	水平方向上是否在父控件的中央
android:centerInVertical	true、false	垂直方向上是否在父控件的中央

续表

属　　性	取　　值	说　　明
android:layout_alignParentTop	true、false	是否与父组件顶部对齐
android:layout_alignParentBottom	true、false	是否与父组件底部对齐
android:layout_alignParentLeft	true、false	是否与父组件左端对齐
android:layout_alignParentRight	true、false	是否与父组件右端对齐
android:layout_alignTop	@id/***	与 id 为***的控件顶部平齐
android:layout_alignBottom	@id/***	与 id 为***的控件底部平齐
android:layout_alignLeft	@id/***	与 id 为***的控件左端平齐
android:layout_alignRight	@id/***	与 id 为***的控件右端平齐
android:layout_above	@id/***	底部和 id 为***的控件顶部平齐
android:layout_below	@id/***	顶部和 id 为***的控件底部平齐
android:layout_toLeftOf	@id/***	右端和 id 为***的控件左端平齐
android:layout_toRightOf	@id/***	左端和 id 为***的控件右端平齐

【例 3-6】 演示 RelativeLayout 的用法

打开 Android Studio，新建一个带空白 Activity 的项目，命名为 RelativeLayout。添加一个 TextView，使其居中，再添加四个 TextView，分别分布在其四个角上。

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="中央"
        android:id="@+id/textView"
        android:layout_centerInParent="true"
        android:textSize="50sp" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="50sp"
        android:text="左上"
```

```
18          android:layout above="@+id/textView"
19          android:layout_toLeftOf="@+id/textView"
20          />
21      <TextView
22          android:layout_width="wrap_content"
23          android:layout height="wrap content"
24          android:textSize="50sp"
25          android:text="右上"
26          android:layout above="@+id/textView"
27          android:layout_toRightOf="@+id/textView" />
28      <TextView
29          android:layout width="wrap content"
30          android:layout height="wrap content"
31          android:textSize="50sp"
32          android:text="左下"
33          android:id="@+id/textView4"
34          android:layout below="@+id/textView"
35          android:layout_toLeftOf="@+id/textView" />
36      <TextView
37          android:layout_width="wrap_content"
38          android:layout height="wrap content"
39          android:textSize="50sp"
40          android:text="右下"
41          android:layout below="@+id/textView"
42          android:layout toRightOf="@+id/textView" />
43  </RelativeLayout>
```

其中，第 6～12 行声明了一个 id 为 textView 的 TextView 控件，第 11 行将其放置在父组件 RelativeLayout 的中央。第 13～42 行分别声明了 4 个 TextView 控件，并分别通过 android:layout_above 和 android:toLeftOf、android:above 和 android:toRightOf、android:layout_below 和 android:toLeftOf、android:layout_below 和 android:toRightOf 的组合放置到 id 为 textView 的组件的左上、右上、左下、右下角。

最终程序运行界面如图 3-8 所示。



图 3-8　RelativeLayout 样式

3.3.4　TableLayout

TableLayout（表格布局）是将布局页面划分为行和列构成的单元格，继而将子元素放置在单元格中的一种布局。该布局组件中用<TableRow>和</TableRow>标记表示一行单元格。与大多数编程语言相似，TableRow 的行数和列数都是从 0 开始计数。此外，Table 中子元素可以不指定 android:layout_height 和 android:layout_width，因为它们默认为单个单元格的高和宽。

TableLayout 的主要全局属性如下：

- android:collapseColumns——隐藏 TableLayout 里面指定的列。若有多列需要隐藏，可用逗号将需要隐藏的列序号隔开。若需要隐藏所有单元格，则将其设置为“*”;
- android:stretchColumns——设置指定的列为可伸展的列，以填满该行中剩下的空白空间。若有多列，则需要设置为可伸展，可用逗号将需要伸展的列序号隔开。
- android:shrinkColumns——设置指定的列为可收缩的列。当可收缩的列太宽时，此列会自动收缩，以免被挤出屏幕。当需要设置多列为可收缩时，可

用逗号隔开需要收缩的列序号。

子元素即单元格的属性如下所示。

- android:layout_column——指定该控件在 TableRow 中占据的列。
- android:layout_span——指定该控件所跨越的列数。

【例 3-7】 演示 TableLayout 的主要用法

打开 Android Studio，新建一个带空白 Activity 的项目，命名为 TableLayout。

打开 activity_main.xml 文件，按下列代码添加一个 4 行 3 列的 TableLayout。

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:shrinkColumns="2">
    <Button
        android:text="单独占据一行"/>
    <TableRow>
        <Button
            android:text="1 行 0 列"/>
        <Button
            android:text="1 行 1 列" />
    </TableRow>
    <TableRow>
        <Button
            android:text="2 行 0 列"/>
        <Button
            android:text="2 行 1 列"/>
        <Button
            android:text="............2 行 2 列..............."/>
    </TableRow>
    <TableRow>
        <Button
            android:text="3 行 0 列"/>
        <Button
            android:text="3 行 2 列"
            android:layout_column="2"
            />
    </TableRow>
</TableLayout>
```

第 5、6 行声明的 Button 并没有放置在任何一个 TableRow 中，因此它会被当作单独的一行，故完全占据了第 0 行。第 7～12 行 TableRow 标签表示这是完整的一行，其中声明的两个 Button 自动按照前后顺序排列，成为第 1 行第 0 列和第 1 行第 1 列。同样，第 13～20 行再次声明一行，即第 2 行。原本第 2 行第 2 列的 Button 空间会由于过大显示到屏幕外，但第 4 行设置了 android:shrinkColumns="2"，即第 2 列会自动收缩适配屏幕大小，所以第 2 行第 2 列的 Button 最终完整显示出来了。第 21～28 行又声明一行，即第 3 行。此行中的第 2 个 Button 原本应该放置在第 1 列，但是第 26 行 android:layout_column="2" 设置了其占据第 2 列，故其最终位置为第 3 行第 2 列。

程序的最终运行界面如图 3-9 所示。



图 3-9　TableLayout 样式

3.3.5　GridLayout

GridLayout（网格布局）是 Android 4.0 以上版本新增加的一种布局，与 TableLayout 大同小异，同样将布局划分为行、列和单元格，并同样支持调整容器中组件的对齐方式。但相比 TableLayout，GridLayout 更为灵活，支持同时控制水平和垂直方向的空间对齐方式，并且支持控件跨行分布。和 TableLayout 一样，GridLayout 对行和列的计数也是从 0 开始的。如果没有指定行数和列数，控件会被默认放在上个控件右边的单元格，若此行已满，则会放在下一行的第 1 个单元格。

GridLayout 的主要全局属性有：

- android:rowCount——设置行数。
- android:columnCount——设置列数。

子元素属性主要有：

- android:layout_gravity——设置控件的对齐方式。
- android:layout_row——设置控件所在行数。
- android:layout_column——设置控件所在列数。
- android:layout_rowSpan——设置控件横跨的行数。
- android:layout_columnSpan——设置空间横跨的列数。

【例 3-8】 演示 GridLayout 的主要用法

打开 Android Studio，新建一个带空白 Activity 的项目，命名为 GridLayout。打开 activity_main.xml 文件，声明一个 3 行×3 列的 GridLayout。

```
      <?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout width="match parent"
    android:layout height="wrap content"
    android:rowCount="3"
    android:columnCount="3">
    <Button
        android:text="1"
        android:layout_gravity="fill"
        android:layout rowSpan="2"
        android:layout_columnWeight="1"/>
    <Button
        android:text="2"
        android:layout gravity="fill"
        android:layout columnSpan="2"
        android:layout columnWeight="1"/>
    <Button
        android:text="3"
        android:layout gravity="fill"
        android:layout_columnSpan="2"
        android:layout column="0"
        android:layout_columnWeight="1"/>
    <Button
        android:text="4"
```

```
25          android:layout_gravity="fill"
26          android:layout_row="1"
27          android:layout_rowSpan="2"
28          android:layout_column="2"
29          android:layout_columnWeight="1"/>
30      <Button
31          android:text="5"
32          android:layout_row="1"
33          android:layout_columnWeight="1"
34          android:layout_column="1"/>
35  </GridLayout>
```

其中第 5 行声明 GridLayout 的行数为 3，第 6 行声明列数为 3。第 7～11 行，声明了一个 Button 控件。没有指定行和列的时候，控件默认按从左到右、从上到下排列，所以该 Button 在第 0 行第 0 列。第 10 行设置 android:layout_rowSpan="2"，即其横跨两行，所以其最终会占据第 0、1 行第 1 列。android:layout_columnWeight 是为了设置列的宽度比例，此处所有 Button 控件的该属性都设为相同的值 1，所以最后单列的宽度相等。其余 Button 控件的设置方法都与第一个相似，此处不再赘述。

最终程序运行的界面如图 3-10 所示。

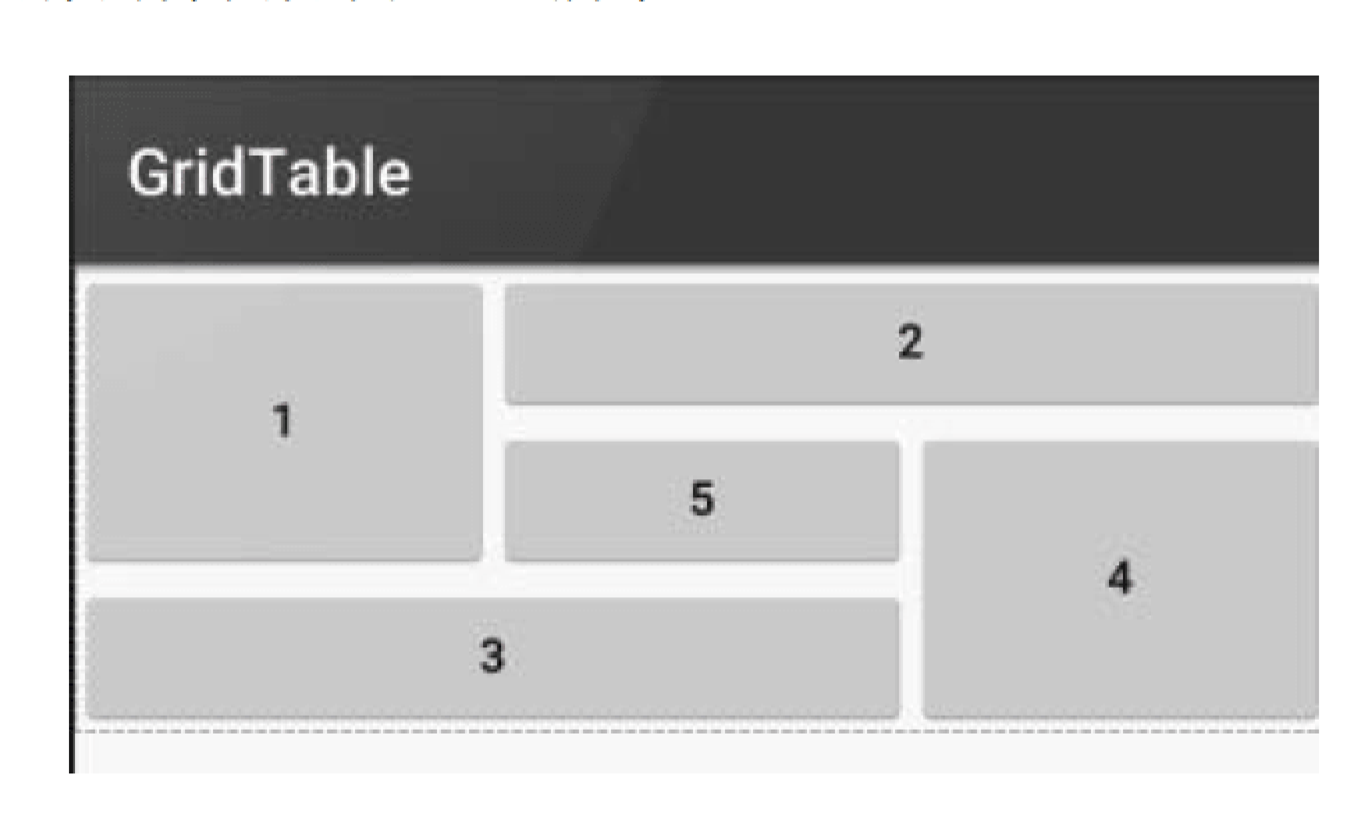


图 3-10　GridTable 样式

3.3.6　Layout 布局小结

除上述详细介绍的常用布局之外，有时可能还会用到其他布局，比如 AbsoluteLayout（绝对布局）——通过指定组件的 x、y 坐标控制组件的位置。在

实际开发中，倘若只用一种布局，往往难以做出美观的界面。因此多数情况下，界面会有多种布局嵌套使用，以达到复杂精巧的预期效果。相同布局可以嵌套，不同布局也可以相互嵌套，非常灵活。

3.4 复合按钮

Android 系统本身封装了多种按钮，除了默认的 Button，常用的还有 Checkbox、RadioButton 和 ToggleButton 三种复合按钮。这三种特殊按钮都继承自类 CompoundButton，而 CompoundButton 类又继承自 Button 类。

3.4.1 CheckBox

CheckBox 有选中与未选中两种状态，默认是未选中状态，其一般形式如图 3-11 所示。其最重要的属性是 Checked，在代码中可以用 isCheck()方法判断。改变该属性的方式有三种。

1. XML 中声明

```
android:checked="false"
```

或者

```
android:checked="true"
```

2. 代码动态改变

```
对象. setChecked(false)
```

或者

```
对象.setChecked(true)
```

或者

```
对象.toggle()
```

3. 用户触摸

Checked 属性的改变会触发 OnCheckedChange 事件，因此可以注册 OnCheckedChangeListener 监听器来监听此事件。



图 3-11　CheckBox

3.4.2　RadioButton

RadioButton 为单选按钮，一般样式如图 3-12 所示。通常需要与 RadioGroup 一同使用。RadioGroup 是可以容纳多个 RadioButton 的容器。每个 RadioGroup 中的 RadioButton 同时只能有一个被选中。不同的 RadioGroup 中的 RadioButton 互不相干，即如果组 A 中有一个被选中了，组 B 中依然可以有一个被选中。大部分场合下，一个 RadioGroup 中至少有两个 RadioButton。大部分场合下，一个 RadioGroup 中的 RadioButton 默认会有一个被选中，并建议将其放在 RadioGroup 中的起始位置。

Checked 属性也是 RadioButton 最重要的属性，与 checkBox 相同，它也可以用三种方式改变 check 属性。而且当 checked 属性改变时，也会触发 OnCheckedChange 事件，因此也可以注册 OnCheckedChangeListener 监听器来监听此事件。



图 3-12　RadioButton

3.4.3　ToggleButton

ToggleButton 允许用户在两种状态之间切换一个设置，比如灯的“开”和“关”，其基本样式如图 3-13 所示。Android 4.0（API level 14）及以上版本还引入了一种全新的复合按钮 Switch，它的功能与 ToggleButton 相同，但样式发生了变化，其样式如图 3-14 所示。



图 3-13　ToggleButton　　　图 3-14　Switch

与 CheckBox 和 RadioButton 相同，ToggleButton 也是 CompoundButton 的子类。Checked 属性也是其最重要的属性。也可以用三种方式改变 check 属性。而且当 checked 属性改变时，也会触发 OnCheckedChange 事件，因此也可以注册 OnCheckedChangeListener 监听器来监听此事件。

习　题　3

1．请编程实现 TextView 类的一个子类，使得它默认指定某些特性，比如文字大小、颜色和样式。

2．请编程实现 View 类的一个子类，使得它具有某些样式（可能需要用到 Android 绘图功能）。

3．请实现一个简易加法计算器，它应该有两个输入和一个输出。请注意界面的美观。

4．请实现一个计算器，它应该有 0～9 十个数字、小数点和加减乘除以及等于、清空、删除一个字符等按钮。

5．请实现一个学生信息录入系统，它应该可以输入姓名、学号、联系方式和选择性别，以及确认是否团员。请注意界面的美观。

第 4 章 Activity 与 Fragment

本章将讲解 Android 中最为重要的概念之一——Activity(活动)的相关知识，你将了解到 Activity 的创建和 Activity 的生命周期。本章也会详细讲解 Fragment(碎片)的相关知识以及 Activity 和 Fragment 之间相互通信的方式。

4.1 Activity 详解

Activity 是一种可以包含用户界面的组件，主要用于和用户进行交互。每个 Activity 都表示一个屏幕，通常一个应用程序至少要包含一个实现应用程序的主 UI 功能的主界面屏幕，这个主界面一般由许多 Fragment 组成，并且通常由一组次要 Activity 支持，要在屏幕间切换就必须要启动一个新的 Activity。

在前几章中，开发者所创建的可以直接成功运行的应用程序中用来显示主屏幕的 MainActivity 都是由 Android Studio 自动生成的。随着学习的进一步深入，应用程序项目将需要更多的 Activity，因此下面将讲解如何自己手动创建一个 Activity。

新建一个 Java Class，类名命名为 MyActivity。每一个新的 Activity 都需要继承自 android.app 包下的 Activity 类并重写它的 onCreate()方法，它的基本框架如下面的代码所示：

```
import android.app.Activity;
import android.os.Bundle;
public class MyActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

Activity 基类封装了窗口显示处理功能，能够显示一个空白屏幕，接下来开发

者可以在上面使用 Fragment、layout（布局）和 view（视图）来创建 UI。在第 3 章你已经学过了布局和视图，视图是用来显示数据和提供用户交互的 UI 控件，继承自 View 基类，布局可以容纳多个视图，继承自 ViewGroup 基类，而 ViewGroup 类本身也派生自 View 类。在本章的后半部分还将讲解 Fragment，它用来封装 UI 的各部分，从而方便地创建动态界面，优化 UI 布局。

要把一个 UI 加载到一个 Activity，需要在 Activity 的 onCreate()方法中调用 setContentView，例如下面的代码会将 TextView 的一个实例加载到当前 Activity 中：

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    TextView textView = new TextView(this);
    setContentView(textView);
}
```

更为一般的情况是把一个布局文件的资源 ID 传入 setContentView 中，例如下面的代码，把 Android Studio 默认生成的 activity_main 加载到 myActivity：

```
import android.app.Activity;
import android.os.Bundle;
public class MyActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout. activity main);
    }
}
```

为了在应用程序中使用一个 Activity，需要在 Manifest 中对其进行注册。在 AndroidManifest 的 application 节点内添加一个新的 activity 标签，没有 activity 标签的活动是不能够显示的，试图显示它们会导致抛出运行时异常。下面是一个典型的 activity 标签的定义：

```
<activity android:name=".MyActivity" >
    <intent-filter>
        <action
```

```
            android:name="android.intent.action.MAIN"
              android:label="This is MyActivity" />
       <category
              android:name="android.intent.category.LAUNCHER" />
     </intent-filter>
 </activity>
```

android:name 指定具体注册的 Activity 名称，由于最外层的<manifest>标签中已经通过 package 属性指定了程序的包名，因此在注册活动时这一部分可以省略，直接使用.MyActivity。

android:label 指定 activity 标题栏的内容，需要注意的是，为主活动指定的 label 不仅会成为标题栏中的内容，还会成为启动器中应用程序显示的名称。

activity 标签中还必须添加 intent-filter 节点，以确定用来启动 Activity 的 Intent。每个 Intent Filter 定义一个或多个 Activity 支持的动作（action）和分类（category）。关于 Intent 和 Intent Filter 的详细内容本书将在第 5 章讲解，现在开发者只需知道：为了让 MyActivity 成为这个应用程序项目的主活动，即在手机上单击应用图标时首先启动的就是这个活动，则它必须包含一个监听 MAIN 动作和 LAUNCHER 分类的 Intent Filter。

4.2 Activity 的生命周期

本书在第 1 章讲解了 Android 应用程序的生命周期，与传统的应用平台不同，Android 应用程序不能控制它们自己的生命周期，应用程序组件必须实时监听应用程序的变化并作出适当的反应。因此，掌握 Android 应用程序的生命周期对 Android 开发者来说非常重要，而一个应用程序的优先级又受其优先级最高的 Activity 的影响，因此接下来本书将讲解 Activity 的生命周期，从而帮助开发者确定其父应用程序的优先级，写出更加连贯流畅的程序，合理管理应用资源，带来更好的用户体验。

4.2.1 Activity 栈

Android 中的 Activity 是可以层叠的，Activity 栈是当前所有正在运行的 Activity 的后进先出的集合，每一个 Activity 的状态由它在栈中所处的位置决定。每启动一个新的 Activity，它就变为运行状态，并被移动到栈顶，覆盖在原 Activity 之上，当用户点击返回键时则会将栈顶的 Activity 出栈并将其下面的 Activity 移

动到栈顶，变为运行状态。

4.2.2 Activity 状态

随着 Activity 的创建和销毁，在栈中的移进移出，每个 Activity 在其生命周期中都会经历下面四种可能的状态。

1. 运行状态

当一个 Activity 位于栈顶，是可见的、具有焦点的前台 Activity 时，即处于运行状态。Android 运行时会根据需要销毁栈下面部分的 Activity 来保证处于运行状态的 Activity 拥有所需要的资源。

2. 暂停状态

当一个 Activity 不再处于栈顶位置，仍然可见但没有获得焦点时，就进入了暂停状态，通常表现为一个透明的或者非全屏的 Activity(例如对话框位于 Activity 前面时)。处于暂停状态的 Activity 可认为近似于处于运行状态，但无法接收用户的输入事件，Android 也不愿意回收这种 Activity，因为它仍然是可见的，回收可见的 Activity 会造成不好的用户体验，因此只有在内存极低的情况下，系统才会考虑回收这种 Activity。

3. 停止状态

当一个 Activity 不再处于栈顶位置，且完全不可见时，即进入了停止状态。此时，Activity 仍然会停留在内存中，保存着相应的状态和成员变量，一旦 Activity 变为运行状态，它就会恢复那些被保存的值。但是这并不可靠，当系统其他地方需要使用内存时，处于停止状态的 Activity 有可能会被回收。

4. 销毁状态

当一个 Activity 从栈中移除后就变为销毁状态。Android 运行时会最先回收处于这种状态的 Activity，从而保证手机内存充足。

4.2.3 Activity 的生存期

为了保证应用程序组件可以对状态改变作出反应，Android 提供了一系列事件处理程序。当 Activity 在完整的、可见的和活动的生存期之间相互转化时，这些回调方法就会被触发。下面首先来讲解 Android 类中的这七个生命周期方法，它们的定义如下：

```
protected void onCreate(Bundle savedInstanceState)
protected void onRestart()
```

```
protected void onStart()
protected void onResume()
protected void onPause()
protected void onStop()
protected void onStop()
```

以上七个方法又可将 Activity 分为三种生存期。

1．完整生存期：onCreate() -> … -> onDestroy()

Activity 在第一次被创建的时候调用 onCreate()方法完成 Activity 的初始化操作并填充 UI，在销毁时调用 onDestroy()方法清理所有的资源，包括结束线程、关闭网络、数据库等外部链接。onCreate()方法和 onDestroy()方法之间所经历的就是 Activity 的完整生存期。

2．可见生存期：onStart() -> … -> onStop()

onStart()方法和 onStop()方法之间所经历的为 Activity 的可见生存期。在可见生存期内，活动对于用户总是可见的，但它们有可能无法和用户进行交互，即可能不具有焦点。开发者可以通过这两个方法合理地管理对用户可见的资源，例如在 onStart()方法中对任何 Activity 可见时所需要改变的 UI 资源进行加载，在 onStop()方法中挂起当 Activity 不可见时不需要的 UI 更新、线程等，从而保证处于停止状态的 Activity 不会占用过多内存。onRestart()方法与 onStart()方法的不同之处在于，onRestart()方法用于实现只有当 Activity 在其完整生存期内重启时才能完成的特殊处理。

3．焦点生存期：onResume() ->… -> onPause()

onResume()方法和 onPause()方法之间所经历的为 Activity 的焦点生存期。在焦点生存期内，Activity 总是可以和用户进行交互。在一个 Activity 从创建到销毁的完整生存期内，它会经历焦点生存期和可见生存期的一次或多次重复。

更上一层 为了保证连贯流畅的用户体验，Activity 从暂停或停止转换为运行状态时用户应该感觉不到任何区别，因此当一个 Activity 被暂停或停止时应该保存所有的 UI 状态以便在 Activity 变为运行状态时恢复。从上述生命周期可以看出，系统在终止应用程序进程时会调用 onPause()、onStop()和 onDestroy()方法，onPause()排在最前面，即 Activity 在失去焦点时就可能终止进程，而 onStop()和 onDestroy()方法可能没有机会执行，因此在 onPause()之前就应该调用 onSaveInstanceState()方法，此方法可将 Activity 中的 UI 状态保存在一个

Bundle 对象中，通过传递给 onCreate()方法和 onRestoreInstanceState()方法实现 UI 状态的恢复。由于 Activity 可能在运行时被意外终止，因此 onCreate()方法需要接受一个在最后一次调用 onSaveInstanceState()方法时所保存的 Bundle 对象，将 UI 恢复成上一次的状态，当然这也可以通过重写 onRestoreInstanceState()方法来实现。

4.3 Activity 启动模式

在默认情况下，当多次启动同一个活动时，系统会创建多个实例并把它们一一放入任务栈中，当按 back（手机上的返回键，不同机型返回键设置略有差异）键时，这些活动会一一回退。但有时我们并不需要重复创建多个实例，所以 Android 提供了启动模式来修改系统的默认行为。目前有四种启动模式：standard（默认模式），就是新进入的活动在任务栈中压在已存在的界面之上；singleTop（栈顶复用模式）；singleTask（栈内复用模式）singleInstance（栈内单例模式）。下面对各个启动模式进行详细介绍。

1．standard

standard 即标准启动模式，也是 Activity 的默认启动模式。在这种模式下启动的 Activity 可以被多次实例化，即在同一个任务中可以存在多个 Activity 的实例，每个实例都会处理一个 Intent 对象。如果 Activity A 的启动模式为 standard，并且 A 已经启动，在 A 中再次启动 Activity A，即调用 startActivity（new Intent（this, A.class）)，会在 A 的上面再次启动一个 A 的实例，即当前的栈中的状态为 A→A。

2．singleTop

如果一个以 singleTop 模式启动的 Activity 的实例已经存在于任务栈的栈顶，那么再启动这个 Activity 时，不会创建新的实例，而是重用位于栈顶的那个实例，并且会调用该实例的 onNewIntent()方法将 Intent 对象传递到这个实例中。举例来说，如果使用此模式，那么在任务栈中栈顶为 C 的情况下，再次打开 C，C 界面的 onCreate()方法和 onStart()方法不会被调用，真正调用时 onPause()–>onNewIntent()–>onResume()。如果以 singleTop 模式启动的 Activity 的一个实例已经存在于任务栈中，但是不在栈顶，那么它的行为和 standard 模式相同，也会创建多个实例。

3．singleTask

如果一个 Activity 的启动模式为 singleTask，那么系统会在打开它的时候查询

所有的任务栈，如果有任务栈包含它，那么把这个任务栈移动到所有栈的首位并清除掉这个栈内从该 Activity 到栈顶的其他 Activity，最后调用它的 onNewIntent()方法。如果没有，那就直接在所需任务栈的栈顶创建该活动的实例。

4．singleInstance

该模式具备 singleTask 模式的所有特性，它与 singleTask 的区别是，这种模式下的 Activity 会单独占用一个 Task 栈，具有全局唯一性，即整个系统中就只有一个实例。由于栈内复用的特性，后续的请求均不会创建新的 Activity 实例，只会把它所在的任务调度到前台，重用这个实例，除非这个特殊的任务栈被销毁。该模式常见的应用场景是呼叫来电界面，除此之外，这种模式的使用情况比较罕见。

系统提供了两种方式来设置一个 Activity 的启动模式，除在 AndroidManifest 文件的 activity 标签中设置 android:launchMode 属性以外，还可以通过 Intent 的 Flag 来设置一个 Activity 的启动模式，下面本书简单介绍一些 Flag。

FLAG_ACTIVITY_NEW_TASK

使用一个新的 Task 来启动一个 Activity，但启动的每个 Activity 都将在一个新的 Task 中。该 Flag 通常用在从 Service 中启动 Activity 的场景，由于 Service 中并不存在 Activity 栈，所以使用该 Flag 来创建一个新的 Activity 栈，并创建新的 Activity 实例。

FLAG_ACTIVITY_SINGLE_TOP

使用 singleTop 模式启动一个 Activity，与指定 android：launchMode=singleTop 效果相同。

FLAG_ACTIVITY_CLEAR_TOP

使用 singleTask 模式来启动一个 Activity，与指定 android：launchMode =singleTask 效果相同。

FLAG_ACTIVITY_NO_HISTORY

Activity 使用这种模式启动 Activity，当该 Activity 启动其他 Activity 后，该 Activity 就消失了，不会保留在 Activity 栈中。

使用 StartActivityForResult 方法启动一个 Activity，然后在 onActivityResult()方法中可以接收到上个页面的回传值，但有可能遇到得不到返回值的情况，那可能是因为 Activity 的 LaunchMode 被设置为 singleTask。Android 5.0 之后，Android 的 LaunchMode 与 StartActivityForResult 的关系发生了一些改变。图 4-1 和图 4-2 展示了两个 Activity A 和 B，由 A 页面跳转到 B 页面时，LaunchMode 与 StartActivityForResult 之间的关系。

A \ B	standard	singleTop	singleTask	singleInstance
standard	✓	✓	×	×
singleTop	✓	✓	×	×
singleTask	✓	✓	×	×
singleInstance	×	×	×	×

Android 5.0之前

图 4-1　Android 5.0 之前启动模式与活动跳转的关系

A \ B	standard	singleTop	singleTask	singleInstance
standard	✓	✓	✓	✓
singleTop	✓	✓	✓	✓
singleTask	✓	✓	✓	✓
singleInstance	✓	✓	✓	✓

Android 5.0之后

图 4-2　Android 5.0 之后启动模式与活动跳转的关系

这是因为 ActivityStackSupervisor 类中的 startActivityUncheckedLocked 方法在 Android 5.0 中进行了修改。在 Android 5.0 之前，当启动一个 Activity 时，系统将首先检查 Activity 的 launchMode，如果将 A 页面设置为 SingleInstance，或者将 B 页面设置为 singleTask 或者 singleInstance，则会在 LaunchFlags 中加入 FLAG_ACTIVITY_NEW_TASK 标志，而如果含有 FLAG_ACTIVITY_NEW_TASK 标志，onActivityResult 将会立即接收到一个 cancel 的信息，而 Android 5.0 之后这个方法做了修改，修改之后即便启动的页面设置 launchMode 为 singleTask 或 singleInstance，onActivityResult 依旧可以正常工作，也就是说，无论设置哪种启动方式，StartActivityForResult 和 onActivityResult()这一组合都是有效的。如果开发者基于 Android 5.0 做相关开发，需要向下兼容。

4.4 Fragment 详解

Fragment（碎片）允许用来将活动拆分成多个独立封装可重用的组件，它能让程序更加合理和充分地利用大屏幕的空间，因而在平板上应用非常广泛。虽然 Fragment 是一个全新的概念，但它和活动十分相似，都可以包含布局，都拥有自己的生命周期，因此本书在此处对其进行讲解，便于开发者将其与 Activity 进行对比学习。

新建一个 MyFragment 类，继承自 Fragment 类。注意，虽然 Android 支持包和 Android 本地开发包都含有 Fragment 的相关类，建议使用 android.app.Fragment，因为我们的程序是面向 Android 5.0 以上系统的，另一个包下的 Fragment 类主要是用于兼容低版本的 Android 系统。

MyFragment 的代码如下所示：

```
public class MyFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater,
        ViewGroup container,Bundle savedInstanceState) {
     return inflater.inflate(R.layout.my fragment,
             container, false);
     //如果该 Fragment 没有 UI，返回 null
    }
}
```

这里仅仅是重写了 Fragment 类的 onCreateView()方法，然后在这个方法中通过 LayoutInflater 的 inflate()方法将定义的 Fragment 对应的布局 my_fragment 动态加载进来。

和 Activity 不同，新建的 Fragment 不需要在 manifest.xml 中进行注册，这是因为 Fragment 必须嵌入到一个 Activity 中才能够存在，而且它的生命周期也依赖于它所嵌入的 Activity。

要把一个 Fragment 添加到一个 Activity 中，最简单的方法是在 Activity 的布局中使用<fragment>标签来添加它。此处以修改之前 activity_main.xml 中的代码为例：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout width="match parent"
    android:layout_height="match_parent">
    <fragment
      android:name="com.example.fragmentTest.MyFragment"
      android:id="@+id/my fragment"
      android:layout_width="match_parent"
      android:layout height="match parent"
      android:layout weight="1"
   />
</LinearLayout>
```

要充分地体现 Fragment 的优势，更好的方式是在程序运行时动态地将 Fragment 添加到 Activity 当中。

首先基于当前的应用程序状态使用容器 View 来创建布局，Fragment 在运行时可以放入到一个容器 View 内，例如下面的代码将 Fragment 放在了一个 FrameLayout 中。在第 3 章中已经讲解了 FrameLayout 布局，这是 Android 中最简单的一种布局，它没有任何的定位方式，所有的控件都会摆放在布局的左上角。由于这里只需要在布局里放入一个 Fragment，因此使用 FrameLayout 非常合适。之后开发者通过在代码中替换 FrameLayout 里的内容即可实现动态添加 Fragment 的功能。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
     android:orientation="horizontal"
    android:layout width="match parent"
    android:layout height="match parent">
     <FrameLayout
       android:id="@+id/ui container"
       android:layout_width="match_parent"
       android:layout height="match parent"
       android:layout_weight="1"/>
</LinearLayout>
```

接下来开发者需要在 Activity 的 onCreate()中使用 Fragment Transaction 来创建相应的 Fragment 并把它加入到对应的父容器中，具体步骤如下：

（1）创建待添加的 Fragment 实例。

```
MyFragment fragment = new MyFragment();
```

（2）获取到 FragmentManager，在活动中可以直接调用 getFragmentManager()方法得到。

```
FragmentManager fragmentManager = getFragmentManager();
```

（3）通过调用 beginTransaction()方法开启一个事务。

```
FragmentTransaction fragmentTransaction =
 fragmentManager.beginTransaction();
```

（4）向容器内添加、替换 Fragment 或删除容器内的 Fragment。

添加一个 Fragment 时需要指定要添加的 Fragment 实例和要放置它的容器 View 的 ID。对于没有 UI 的 Fragment，它不应该与一个容器 View 关联，必须指定一个 Tag 来标识该 Fragment：

```
//有 UI 的 Fragment
fragmentTransaction.add(R.id.ui_container,fragment);
//无 UI 的 Fragment
fragmentTransaction.add(fragment, MY_FRAGMENT_TAG);
```

把一个 Fragment 替换为另一个 Fragment 需要使用 replace()方法，指定要替换的 Fragment 的父容器的 ID、替换它的新 Fragment 和新 Fragment 的 Tag 标识（可选）：

```
fragmentTransaction.replace(R.id.ui_container,new MyFragment());
```

要删除一个 Fragment 首先要使用 findFragmentById()或 findFragmentByTag()方法获得对该 Fragment 的引用，然后把其实例作为参数传给 remove()方法：

```
Fragment removeFragment =
fragmentManager.findFragmentById(R.id.ui container);
fragmentTransaction.remove(removeFragment);
```

（5）将事务添加到返回栈。

碎片与活动类似，用户自然也希望使用 Fragment 创建的布局能够通过 Back 键返回到上一布局，因此 FragmentTransaction 中提供了一个 addToBackStack()方法，可以在调用 commit 方法之前将一个事务添加到返回栈中， 它接收一个名字用于描述返回栈的状态，一般传入 null 即可。

```
fragmentTransaction.addToBackStack(tag);
```

调用 addToBackStack()方法后，当用户按下 Back 键，之前的事务将会回滚并将 UI 返回到之前的布局。

（6）调用 commit()方法提交事务。

```
fragmentTransaction.commit();
```

4.5 Fragment 的生命周期

Fragment 的生命周期主要取决于其父活动的生命周期，同时当包含它的活动处于某些状态时，添加或删除一个 Fragment 也会影响它自己的生命周期。

4.5.1 Fragment 的状态

与 Activity 类似，Fragment 的生命周期中也会经历下面四种状态。

1．运行状态

当一个 Fragment 是可见的，并且它所关联的 Activity 正处于运行状态时，该 Fragment 也处于运行状态。

2．暂停状态

当一个 Activity 进入暂停状态时，与它相关联的可见 Fragment 进入到暂停状态。

3．停止状态

当一个 Activity 进入停止状态时，与它相关联的 Fragment 进入到停止状态。

或者，通过调用 FragmentTransaction 的 remove()、replace()方法将 Fragment 从 Activity 中移除且又在事务提交之前调用 addToBackStack()方法，这时 Fragment 也会进入到停止状态。进入停止状态的 Fragment 对用户完全不可见且有可能被系统回收。

4．销毁状态

Fragment 总是依附于 Activity 而存在的，因此当 Activity 被销毁时，与它相

关联的 Fragment 就会进入销毁状态。

或者，通过调用 FragmentTransaction 的 remove()、replace()方法将 Fragment 从 Activity 中移除，但在事务提交之前并没有调用 addToBackStack()方法，这时 Fragment 也会进入到销毁状态。

4.5.2 Fragment 的生命周期方法

Android 为 Fragment 提供了一系列与 Activity 相似的事件处理程序，当 Fragment 被创建、启动、恢复、暂停、停止和销毁时，这些回调方法就会被触发。除此之外，Fragment 还包含一些附加的回调方法用来实现 Fragment 与 Activity 的绑定和分离、Fragment 用户界面的创建和销毁以及与 Fragment 相关联的 Activity 的创建过程的完成情况。这些覆盖 Fragment 生命周期各个环节的回调方法定义如下：

```
protected void onAttach(Activity activity)
protected void onCreate(Bundle savedInstanceState)
protected View onCreateView(LayoutInflater inflater,
ViewGroup container,Bundle savedInstanceState)
Protected void onActivityCreated(Bundle
savedInstanceState)
protected void onRestart()
protected void onStart()
protected void onResume()
protected void onPause()
protected void onDestroyView()
protected void onDestroy()
protected void onDetach()
```

下面对 Fragment 特有的三类生命周期方法进行进一步解释。

1．Fragment 与 Activity 的绑定和分离

Fragment 的完整生存期开始于调用 onAttach()方法将其与某 Activity 绑定，结束于调用 onDetach()方法将其与 Activity 分离。由于 onAttach()会在碎片自身或与它相关联的 Activity 完成它们的初始化之前被触发，因此 onAttach()方法主要用于获取与碎片相关联的 Activity 的引用，为进一步的初始化做准备。

2．Fragment 用户界面的创建和销毁

与 Activity 不同的是，Fragment 的 UI 不在 onCreate()方法中进行初始化，而是在 onCreateView()方法中完成 UI 的创建和填充操作，onCreateView()方法会获

取它所包含的 View 引用并将其返回。如果该 Fragment 没有 UI，（没有 UI 的 Fragment 通常用来完成定期与 UI 交互的后台任务或者在因配置改变而导致的 Activity 重启的场合提供相关状态的保存），则该方法返回 null。

与 onCreateView()方法相对应，Android 提供 onDestroyView()方法用于清除资源相关的 View，完成 Fragment 用户界面的销毁。

3．onActivityCreated()

如果 Fragment 要同与它相关联的 Activity 进行交互，则需要一直等到 Fragment 所在的 Activity 已完成了初始化且它的 UI 已经完全构建完成，此时 onActivityCreated 事件被触发。因此，对于那些只有在与 Fragment 相关联的 Activity 初始化完成后或者 Fragment 的 View 被完全填充后才能做的事情，可以重写该方法来实现。

在 Fragment 中同样可以通过 onSaveInstanceState()方法来保存数据，因为进入停止状态的 Fragment 极有可能在系统内存不足时被回收。保存下来的状态在 onCreate()、onCreateView()和 onActivityCreated()三个方法中都可以重新得到，它们都包含一个 Bundle 类型的 savedInstanceState 参数。

4.6 Fragment 与 Activity 间通信

一个 Fragment 的实例总是和包含它的 Activity 直接相关。Fragment 可以通过 getActivity()方法来获得 Activity 的实例，然后就可以调用一些方法（例如 findViewById()）。如：

```
View listView = getActivity().findViewById(R.id.list);
```

但是调用 getActivity()时，Fragment 必须和 Activity 关联，否则将会返回一个空指针 null。

类似的，Activity 也可以获得一个 Fragment 的引用，从而调用 Fragment 中的方法。获得碎片的引用要用 FragmentManager，之后可以调用 findFragmentById() 或者 findFragmentByTag()，比如：

```
ExampleFragment fragment = (ExampleFragment)
getFragmentManager().findFragmentById(R.id.example_fragment);
```

有时候，可能需要 Fragment 和 Activity 共享事件，一个比较好的做法是在 Fragment 里面定义一个回调接口，然后要求宿主 Activity 实现它。

当 Activity 通过这个接口接收到一个回调，它可以同布局中的其他 Fragment

分享这个信息。例如，一个新闻显示应用在一个 Activity 中有两个 Fragment——FragmentA 显示文章题目的列表，FragmentB 显示文章。

所以当一个文章被选择的时候，FragmentA 必须通知 Activity，然后 Activity 通知 FragmentB，让它显示这篇文章。

这个情况下，在 ActivityA 中声明一个这样的接口 OnArticleSelectedListener：

```
public static class FragmentA extends ListFragment {
    public interface OnArticleSelectedListener {
        public void onArticleSelected(Uri articleUri);
    }
    ...
}
```

之后包含这个 Fragment 的活动实现这个 OnArticleSelectedListener 接口，用重写的 onArticleSelected()方法将 FragmentA 中发生的事通知 FragmentB。

为了确保宿主 Activity 实现这个接口，在 FragmentA 的 onAttach()方法中通过对传入的 Activity 进行强制类型转换，实例化一个 OnArticleSelectedListener 对象：

```
public static class FragmentA extends ListFragment {
    OnArticleSelectedListener mListener;
    ...
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        try {
            mListener = (OnArticleSelectedListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException("活动未实现碎片接口");
        }
    }
    ...
}
```

如果 Activity 没有实现这个接口，那么 Fragment 将会抛出异常。如果成功了，mListener 将会是宿主 Activity 实现 OnArticleSelectedListener 接口的一个引用，所以通过调用 OnArticleSelectedListener 接口的方法，Fragment A 可以和 Activity 共享事件。

比如，如果 FragmentA 是 ListFragment 的子类。每一次用户单击一个列表项目时，系统会调用 Fragment 中的 onListItemClick() 方法，在这个方法中可以调用 onArticleSelected()方法与宿主 Activity 共享事件。

```
public static class FragmentA extends ListFragment {
    OnArticleSelectedListener mListener;
    ...
    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        Uri noteUri =
ContentUris.withAppendedId(ArticleColumns.CONTENT URI, id);
        mListener.onArticleSelected(noteUri);
    }
    ...
}
```

最后需要在宿主 Activity 中实现该接口，并通知 Fragment B 即可。

习　题　4

1. 请简述 Activity 的生命周期。
2. 请简述 Fragment 的生命周期。
3. 请简述 Activity 和 Fragment 的异同。
4. 如果需要在 Fragment 和 Activity 绑定时进行某些操作，该怎么办？
5. 为了在应用程序的 Activity 主进程被自动杀死时保存关键数据，应该怎么办？
6. 请手动实现两个 Fragment 的通信。

第 5 章　高级 UI

第 4 章介绍了安卓 UI 的一些基本知识。这些知识构成了美轮美奂的安卓大厦的地基。但想要实现更多功能，还需要进一步地研读。因此本章介绍了一些高级控件和其他深层次的内容，以实现更多功能。

5.1　Toast 和 Dialog

从严格意义上说，Toast 和 Dialog 都不属于 UI 组件，因为它们都不是 View 类的子类，也不能在 xml 文件中声明，但鉴于它们在提醒用户、与用户交互等方面的巨大作用，本章还是对其略作介绍。

5.1.1　Toast

Toast 是 Android 系统中用来显示信息的一种机制，是一种简易的消息提示框。由于它的设计初衷是尽可能不引发用户的注意，同时还能向用户显示信息，因此它没有焦点，而且会在一段时间后自动消失。使用 Toast 需要引入包：android.widget.Toast。

声明一个 Toast，可以使用 Toast 的静态函数 makeText（Context context, CharSequence text, int duration），其中 context 的类型为 Context，为上下文环境；text 为需要显示的文字信息；duration 的类型为 int，指持续时间，一般用 LENGTH_LONG（长）和 LENGTH_SHORT（短）表示。Toast 可以自定义位置，使用类的成员函数 setGravity（int gravity, int xOffset, int yOffset），三个参数都为 int 类型。第一个参数设置 toast 在屏幕中显示的位置；第二个参数表示相对于第一个参数设置 toast 位置在水平方向的偏移量，正数向右偏移，负数向左偏移；第三个参数则表示在垂直方向上的偏移量。如果不自定义位置，则 Toast 默认会在屏幕的下端中间显示。要使 Toast 显示出来，需调用成员函数 show()。

此外，Toast 还可以利用 getView()得到 Toast 所在的视图（View）后再用 addView（View child）添加 UI 组件，或者直接用 setView（View view）完全自定

义显示方式。

【例 5-1】 演示 Toast 的用法

打开 Android Studio，新建一个带空白 Activity 的项目，命名为 ToastDemo。

打开 activity_main.xml 文件，添加三个 Button，以触发显示 Toast。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center_horizontal">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="默认显示"
        android:id="@+id/button" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="自定义位置"
        android:id="@+id/button2" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="带图片显示"
        android:id="@+id/button3" />
</LinearLayout>
```

打开 MainActivity.java，创建三种 Toast，分别设置为默认、自定义位置和带图片显示。

```
package com.example.toastdemo;
import …;
public class MainActivity extends AppCompatActivity implements View.OnClickListener{
    Button button1,button2,button3;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity main);
        button1 = (Button)findViewById(R.id.button);
        button2 = (Button)findViewById(R.id.button2);
        button3 = (Button)findViewById(R.id.button3);
        button1.setOnClickListener(this);
        button2.setOnClickListener(this);
        button3.setOnClickListener(this);
    }
    @Override
    public void onClick(View v) {
        if(v==button1){//显示默认样式
            Toast.makeText(getApplicationContext(), "默认样式
" ,Toast.LENGTH LONG).show();
        }else if(v==button2){//自定义位置显示
            Toast toast=Toast.makeText(getApplicationContext(),"自定
           义位置",
                  Toast.LENGTH SHORT);
            toast.setGravity(Gravity.CENTER,0,0);//设置位置为正中央
            toast.show();//显示 toast
        }else if(v==button3){//带图片自定义位置显示
            Toast toast=Toast.makeText(getApplicationContext(),"带
            图片显示
",Toast.LENGTH LONG);
            toast.setGravity(Gravity.CENTER,0,0);//设置位置为正中央
            ImageView imageView = new ImageView(getApplication
             Context());
            imageView.setImageResource(R.mipmap.ic_launcher);
            LinearLayout view=(LinearLayout)toast.getView();
                                                //获取 Toast 所在的视图
            view.addView(imageView);//在 toast 的 view 中添加 imageView
            toast.show();
        }
}
}
```

上述代码中，第 20、21 行代码中直接调用了 Toast 的静态方法 makeText (getApplicationContext()，"默认样式" ,Toast.LENGTH_LONG)。最终效果是在移动设备屏幕中下方出现一个文本提示框，如图 5-1（a）所示。第 23～26 行声明了一个 Toast 对象 toast，并用其成员函数 setGravity(Gravity.CENTER,0,0)，设置 toast 的显示位置为屏幕正中央。最终显示效果如图 5-1（b）所示。第 28～36 行同样声明了一个 Toast 对象 toast，此外还声明了 ImageView 的对象 imageView。第 33 行用 toast 的成员函数 getView()获取了 toast 的视图赋予 view，再调用 view 的成员函数 addView(View child)添加了 imageView。其最终显示效果如图 5-1（c）所示。



（a）　　（b）　　（c）

图 5-1　Toast 样式

5.1.2　Dialog

Dialog（对话框）是一个独立存在的容器，只占据屏幕一部分，有自己的标题和边框。当需要提示用户某种重要信息或立刻做出某种操作时，Dialog 便显得格外重要。虽然 Dialog 是所有对话框类的父类，但是谷歌官方并不推荐直接实例化 Dialog，而是提供了以下三种对话框类。

- AlertDialog：消息对话框。提供了诸多成员函数，可以包含标题、图标、提示信息、默认至多三个按钮，也可以支持普通列表、单选列表和多选列表甚至自定义布局。
- DatePickerDialog：日期选择对话框。

- TimePickerDialog：时间选择对话框。

由于后两种对话框比较简单，本章只介绍 AlertDialog。

AlertDialog 的常用方法如下：

- AlertDialog.Builder(Context)——对话框构造器 Builder 的构造方法。
- create()——构造对话框。
- show()——显示对话框。
- dismiss()——关闭对话框。
- setTitle()——设置对话框的标题。
- setIcon()——设置对话框的图标。
- setMessage()——设置对话框的提示内容。
- setItems()——设置普通列表。
- setSingleChoiceItems()——设置单选列表。
- setMultiChoiceItems()——设置多选列表。
- setPositiveButton()——设置确认按钮。
- setNegativeButton()——设置取消按钮。
- setNeutralButton()——设置忽略或稍后再提示按钮。
- setView()——设置自定义视图。

创建 AlertDialog 实例需要使用 AlertDialog 的内部类 Builder。构造一个 AlertDialog 需要如下步骤：

（1）实例化 AlertDialog.Builder。

```
AlertDialog.Builder builder = new
AlertDialog.Builder(getActivity());
```

（2）利用 AlertDialog.Builder 的诸多 set 方法设置属性。

（3）调用 AlertDialog.Builder 的 create 方法得到 AlertDialog 的实例。

```
AlertDialog dialog = builder.create();
```

然后调用 dialog.show()显示对话框。这两部还可以化简为使用 builder.show()直接显示对话框。

如果对话框需要显示自定义视图，可以使用 LayoutInflater 的静态方法 from(Context)获取 LayoutInflater 实例，再用其成员函数 inflate(int resource, ViewGroup root)从布局文件中得到自定义视图。

【例 5-2】 演示普通对话框、列表对话框和自定义对话框的使用方法

打开 Android Studio，新建一个带空白 Activity 的项目，命名为 Dialog。打开 activity_main.xml 文件，添加三个按钮，以分别触发三种对话框的显示。

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical">
    <Button
        android:text="普通对话框"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button1"
        />
    <Button
        android:text="列表对话框"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button2"
        android:layout_marginTop="20dp"
        android:layout_marginBottom="20dp" />
    <Button
        android:text="自定义对话框"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button3"
        />
</LinearLayout>
```

创建自定义对话框的布局文件 dialog.xml，添加两个 EditText 以分别输入用户名和密码、一个登录按钮和一个取消按钮。xml 文件内容如下：

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp">
```

```
    <EditText
        android:layout width="match parent"
        android:layout height="wrap content"
        android:hint="用户名"
        android:id="@+id/nameEdit" />
    <EditText
        android:layout width="match parent"
        android:layout_height="wrap_content"
        android:inputType="textPassword"
        android:hint="密码"
        android:id="@+id/pwdEdit" />
    <LinearLayout
        android:orientation="horizontal"
        android:layout width="match parent"
        android:layout_height="wrap_content">
        <Button
            android:text="取消"
            android:layout width="0dp"
            android:layout_height="wrap_content"
            android:id="@+id/cancelButton"
            android:layout weight="1" />

        <Button
            android:text="登录"
            android:layout width="0dp"
            android:layout height="wrap content"
            android:id="@+id/loginButton"
            android:layout_weight="1" />
    </LinearLayout>
</LinearLayout>
```

打开 MainActivity.java，修改如下：

```
package com.example.wb.dialogdemo;
import …;

public class MainActivity extends AppCompatActivity implements
 View.OnClickListener{
```

```
    Button ordinaryDialogBtn, listDialogBtn, customDialogBtn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ordinaryDialogBtn = (Button)findViewById(R.id.button1);
        listDialogBtn = (Button)findViewById(R.id.button2);
        customDialogBtn = (Button)findViewById(R.id.button3);
        ordinaryDialogBtn.setOnClickListener(this);
        listDialogBtn.setOnClickListener(this);
        customDialogBtn.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        if(v==findViewById(R.id.button1)){
            AlertDialog.Builder dialog = new AlertDialog.Builder(this);
            dialog.setTitle("普通对话框");
            dialog.setMessage("这是一本优秀的安卓教材");
            dialog.setPositiveButton("确定", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog,int which) {
                    dialog.dismiss();
                }
            });
            dialog.show();
        }
        else if(v==findViewById(R.id.button2)){
            final String[] colors=new String[]{"红色","黄色","橙色","蓝色","绿色"};
            AlertDialog.Builder dialog = new AlertDialog.Builder(this);
            dialog.setTitle("列表对话框");
            dialog.setItems(colors,new DialogInterface.OnClickListener(){
                @Override
                public void onClick(DialogInterface dialog, int which) {
```

```
40                      Toast.makeText(getApplicationContext(),colors
41  [which]+"被点击",Toast.LENGTH_LONG).show();
42                  }
43              });
44              dialog.show();
45          }
46          else if(v==findViewById(R.id.button3)){
47              AlertDialog.Builder dialog = new AlertDialog.Builder (this);
48              final View dialogView =
49  LayoutInflater.from(this).inflate(R.layout.dialog,null);
50              dialog.setTitle("自定义对话框");
51              dialog.setView(dialogView);
52              dialog.show();
53          }
54      }
55  }
```

其中第 21～31 行创建了一个普通对话框，第 34～44 行创建了一个列表对话框，第 47～52 行创建了一个自定义对话框。为了节省篇幅，此处并没有添加自定义视图中的事件处理，但其原理与 Activity 的事件处理相同。三种对话框效果如图 5-2 所示。



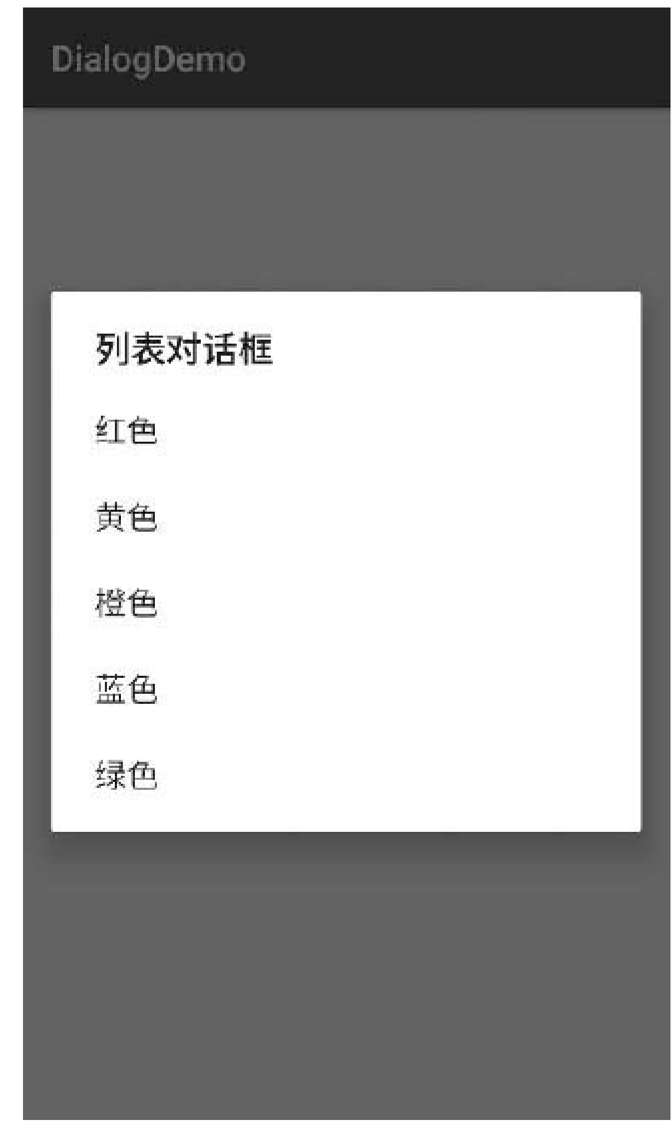




图 5-2　对话框

5.2　Spinner

Spinner 为下拉选择框，提供了从一个数据集合中快速选择一个值的途径。默认情况下 Spinner 显示的是当前选择的值，单击 Spinner 会弹出一个包含所有可选值的下拉菜单或者弹出框，从该菜单中可以为 Spinner 选择一个新值。默认弹出形式与主题有关，也可用通过在 xml 文件中设置 android:spinnerMode="dropdown" 或 android:spinnerMode="dialog"指定弹出形式，但 Android 2.3 中没有 spinnerMode 属性，系统默认将弹出菜单显示为 dialog。

Spinner 的选择项可以直接在 xml 文件中通过 entries 属性设置，但一般是 Java 文件中通过 adapter（适配器）来与 Spinner 绑定数据。Spinner 绑定适配器需要用 setAdapter(Adapter adapter)方法。响应 Spinner 选择事件时可以通过 OnItemSelectedListener 的回调方法实现。

【例 5-3】 演示 Spinner 的用法

打开 Android Studio，新建一个带空白 Activity 的项目，命名为 SpinnerDemo。打开 activity_main.xml 文件，添加 3 个 Spinner，分别用于演示 xml 绑定数据源、默认 adapter 绑定数据源和自定义 adapter 绑定数据源 3 种方法，再添加 3 个 TextView 用于显示对应 Spinner 的选择结果。xml 文件如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
    android:layout_marginLeft="20dp"
    android:layout_marginRight="20dp">
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <Spinner
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:id="@+id/spinner1"
```

```
            android:layout_weight="1"
            android:entries="@array/cities"
            android:gravity="center" />
        <TextView
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:id="@+id/textView1"
            android:layout_weight="1"
            android:gravity="center" />
    </LinearLayout>
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="50dp"
        android:layout_marginTop="50dp">
        <Spinner
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:id="@+id/spinner2"
            android:layout_weight="1"
            android:gravity="center" />
        <TextView
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:id="@+id/textView2"
            android:layout_weight="1"
            android:gravity="center" />
    </LinearLayout>
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <Spinner
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:id="@+id/spinner3"
            android:layout_weight="1"
```

```
55              android:gravity="center" />
56          <TextView
57              android:layout width="0dp"
58              android:layout_height="match_parent"
59              android:id="@+id/textView3"
60              android:layout_weight="1"
61              android:gravity="center" />
62      </LinearLayout>
63  </LinearLayout>
```

其中，spinner1 用于演示 xml 绑定数据源，第 18 行指定了它的数据源是 array 资源中的 cities 数组，因此需要修改 values 目录下的 arrays.xml（若没有此文件，则新建之）。修改如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="cities">
        <item>北京</item>
        <item>上海 </item>
        <item>广州</item>
        <item>深圳</item>
    </string-array>
</resources>
```

自定义 Adapter 需要新建一个布局文件，用于指定 Spinner 下拉菜单中每一行的布局。因此在 layout 目录下新建 spinnerview.xml 文件。修改代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout height="wrap content">
    <ImageView
        android:layout width="wrap content"
        android:layout_height="wrap_content"
        app:srcCompat="@mipmap/ic launcher"
        android:id="@+id/imageView"
        android:layout_weight="1" />
```

```
    <TextView
        android:text="TextView"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:id="@+id/textView1"
        android:layout_weight="1"
        android:gravity="center" />

    <TextView
        android:text="TextView"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:id="@+id/textView2"
        android:layout_weight="1"
        android:gravity="center" />
</LinearLayout>
```

其中 textView1 显示省份名，textView2 显示城市名。

为了封装省份名和城市名，需要新建类 City。该类代码只有两个成员变量：province 和 city。代码如下：

```
package com.example.spinnerdemo;
public class City {
    String province,city;
    City(String provinceName, String cityName){
        province = provinceName;
        city = cityName;
    }
    public String getProvince() {
        return province;
    }
    public void setProvince(String province) {
        this.province = province;
    }
    public String getCity() {
        return city;
    }
    public void setCity(String city) {
```

```
        this.city = city;
    }
}
```

自定义 Adapter，需要新建一个 BaseAdapter 的子类。因此新建文件 MyAdapter.java，且令其继承 BaseAdapter。继承该类，需要重写以下四个方法。

（1）int getCount()——返回适配器绑定的数据源的数据条数。

（2）Object getItem(int position)——返回适配器绑定的数据源的第 position 条数据。

（3）long getItemId(int position)——返回适配器绑定的数据源的第 position 条数据的 ID，一般直接返回 position 即可。

（4）View getView(int position, View convertView, ViewGroup parent)——在该函数中绑定自定义视图，并返回之。

MyAdapter 类具体的代码如下：

```
package com.example.wb.spinnerdemo;
import …;
public class MyAdapter extends BaseAdapter {
    private List<City> mList;
    private Context mContext
    public MyAdapter(Context pContext, List<City> cities) {
        this.mContext = pContext;
        this.mList = cities;
    }
    @Override
    public int getCount() {
        return mList.size();
    }
    @Override
    public Object getItem(int position) {
        return mList.get(position);
    }
    @Override
    public long getItemId(int position) {
        return position;
    }
    @Override
```

```
    public View getView(int position,View convertView,ViewGroup parent) {
        LayoutInflater layoutInflater=LayoutInflater.from
        (mContext);
        convertView=layoutInflater.inflater(R.layout.
        spinnerview,null);
        if(convertView != null) {
            TextView textView1
= (TextView)convertView.findViewById(R.id.textView1);
            TextView textView2
= (TextView)convertView.findViewById(R.id.textView2);
            textView1.setText(mList.get(position).getProvince());
            textView2.setText(mList.get(position).getCity());
        }
        return convertView;
    }
}
```

修改 MainActivity.java 文件如下：

```
package com.example.spinnerdemo;
import …;
public class MainActivity extends AppCompatActivity implements
                AdapterView.OnItemSelectedListener{
    Spinner spinner1, spinner2, spinner3;
    TextView textView1, textView2, textView3;
    ArrayAdapter<String> defaultAdapter;
    MyAdapter myAdapter;
    String[] cities1 = new String[]{"杭州","长沙","成都","武汉"};
    City[] cities2 = new City[]{new City("浙江","杭州"),new City
    ("湖南","长沙"),
            new City("四川","成都"),new City("湖北","武汉")};
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        spinner1 = (Spinner)findViewById(R.id.spinner1);
        spinner2 = (Spinner)findViewById(R.id.spinner2);
        spinner3 = (Spinner)findViewById(R.id.spinner3);
```

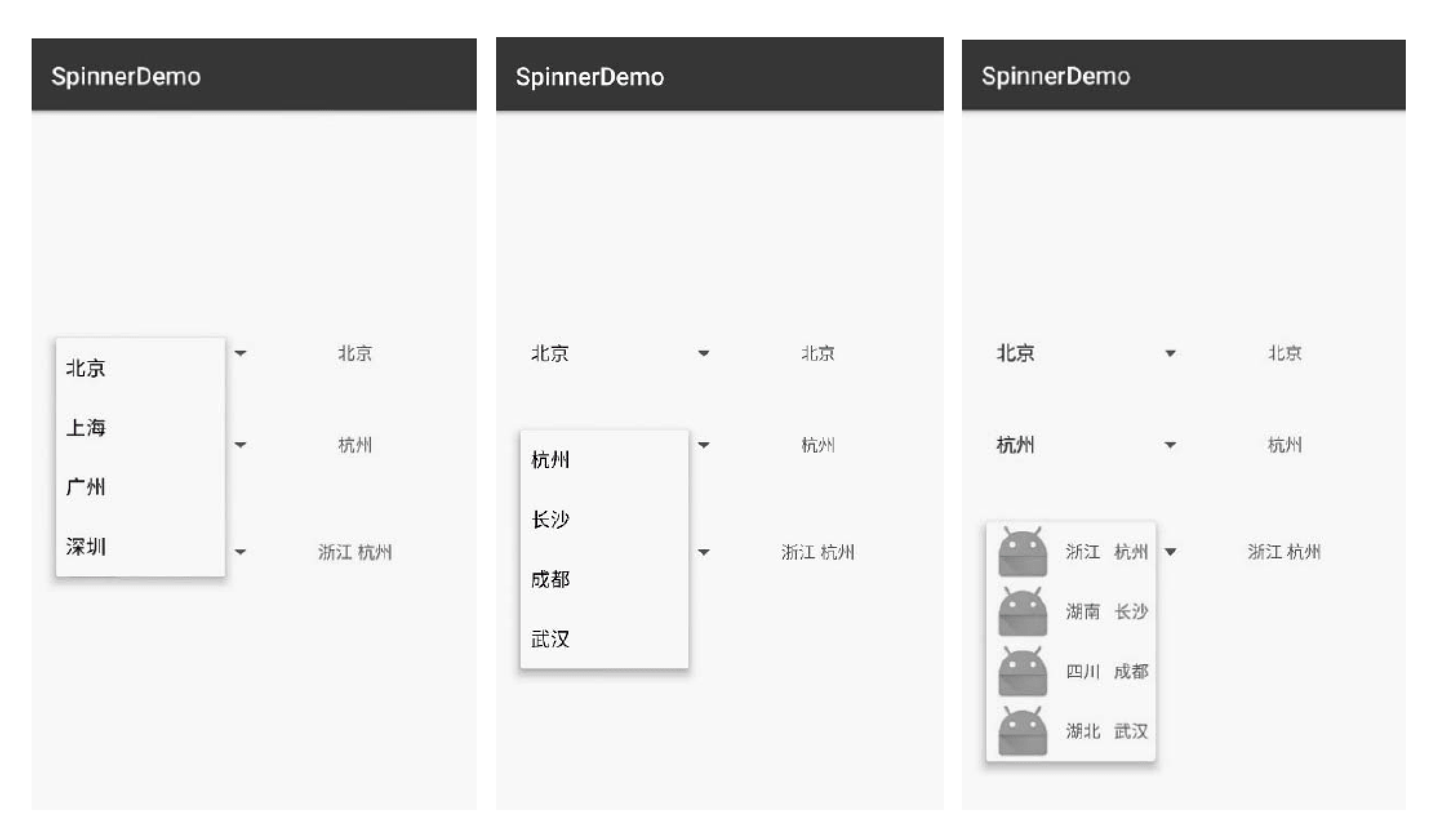
```
19          textView1 = (TextView)findViewById(R.id.textView1);
20          textView2 = (TextView)findViewById(R.id.textView2);
21          textView3 = (TextView)findViewById(R.id.textView3);
22          //使用默认 Adapter 与 spinner2 绑定
23          defaultAdapter = new
24          ArrayAdapter<String>(this,android.R.layout.simple_
            spinner  item, cities1);
25          defaultAdapter.setDropDownViewResource(android.R.layout.
26          simple spinner dropdown item);
27          spinner2 .setAdapter(defaultAdapter);
28          //使用自定义 Adapter 与 spinner3 绑定
29          myAdapter = new MyAdapter(this, Arrays.asList(cities2));
30          spinner3.setAdapter(myAdapter);
31          spinner1.setOnItemSelectedListener(this);
32          spinner2.setOnItemSelectedListener(this);
33          spinner3.setOnItemSelectedListener(this);
34      }
35      @Override
36      public void onItemSelected(AdapterView<?> parent, View view, int position,
37  long id) {
38          if(parent==spinner1){
39              String[] cities =getResources().getStringArray(R.array.
40              cities);
                textView1.setText(cities[position]);
41          }else if(parent==spinner2){
42             textView2.setText(cities1[position]);
43          }else  if(parent==spinner3){
44              textView3.setText(cities2[position].getProvince()+""
45                     +cities2[position].getCity());
46          }
47      }
48      @Override
49      public void onNothingSelected(AdapterView<?> parent) {
50      }
51  }
```

其中第 23、24 行用 ArrayAdapter<T>(Context context, @LayoutRes int resource,

@NonNull T[] objects)构造函数声明了一个 ArrayAdapter<String>类型的适配器。resource 代表 Spinner 未展开菜单时 Spinner 的默认样式，而 android.R.layout.simple_spinner_item 是系统自带的内置布局。第 25、26 行设置的是展开时下拉菜单的样式。android.R.layout.simple_spinner_dropdown_item 也是系统自带的内置布局。

第 29 行声明了一个类型为 MyAdapter 适配器，通过 spinner3.setAdapter 与 spinner3 绑定。

图 5-3 为运行界面截图。



（a）使用 xml 绑定　　（b）使用默认 Adapter 绑定　　（c）使用自定义 Adapter 绑定

图 5-3　运行界面

5.3　ListView

ListView 组件在大部分应用程序中都不可或缺，例如 QQ 的联系人列表、微信的朋友圈列表都用到了 ListView 组件。ListView 主要是显示列表数据，同时可以滚动查看，其样式如图 5-4 所示。

ListView 和 Spinner 都是 AdapterView 的间接子类，因此有很多相同的性质。例如，同样可以用以下三种方式绑定数据源：

图 5-4　ListView 样式

更上一层　除了 ArrayAdapter 类型的适配器，Android 还为开发者封装了多种适配器，比如 SimpleCursorAdapter 和 SimpleAdapter。ArrayAdapter 是最简单的 ListView 的适配器，内部只有一个 TextView。SimpleCursorAdapter 允许你将一个游标的列绑定到 ListView 上，并使用自定义的 layout 显示每个项目，因此可以方便地把数据库的内容以列表的形式展示出来。SimpleAdapter 是向 ListView 添加复杂项的适配器，但 SimpleAdapter 只支持 3 种组件：实现 Checkable 接口的组件类、TextView 类及其子类、ImageView 类及其子类。

（1）在 xml 文件中，指定 android:entries 属性绑定数据源。

（2）在 java 文件中，使用官方自带的 Adapter 绑定数据源，例如：ArrayAdapter、SimpleCursorAdapter、SimpleAdapter。

（3）自定义适配器，使其继承 BaseAdapter 以绑定数据源。

具体的绑定方式请参考【例 5-3】，此处不再赘述。

5.4 Menu

Menu（菜单）是大部分应用呈现都会用到的重要组件。Android 提供了以下三种菜单：Options Menu（选项菜单）、Context Menu（上下文菜单）和 Popup Menu（弹出式菜单）。

1. Options Menu（选项菜单）

默认显示在操作栏的右边，即屏幕的右上方。菜单选项可以以 Action 按钮的样式显示在 ActionBar 上，也可以隐藏在溢出菜单（overflow menu）中单击后再显示出来。

更上一层 Android 3.0 （API level 11）以前，Android 官方的建议是通过触摸 Menu 实体键从屏幕下方弹出选项菜单，但 Android 3.0，官方的建议是使用操作栏显示菜单选项，如图 5-5 所示。



图 5-5 操作栏和选项菜单（1，2，3 所在容器为 App Bar，其中 1 为图标和标题；2 为 Action 按钮；3 为 Overflow Menu 按钮，单击后会弹出其他 Menu 选项）

实现 Options Menu 需要重写 Activity 以下两个方法。

（1）onCreateOptionsMenu(Menu menu)（Fragment 对应 onCreateOptionsMenu()回调）：启动 Activity 时会调用 onCreateOptionsMenu()方法，因此可以在该方法中将菜单资源（使用 XML 定义）注入回调方法的 Menu 中，也可以使用代码手动添加菜单资源。

（2）onOptionsItemSelected(MenuItem menuItem): Options Menu 的选项被单击时，系统会自动调用此方法，开发者可以通过判断 menuItem.getItemId()方法判断是哪个选项被单击。

发生事件时，如果需要执行菜单更新，则必须调用 invalidateOptionsMenu()来请求系统调用 onPrepareOptionsMenu()。在 onPrepareOptionsMenu()方法中通过 menu.add() 等操作修改菜单项。

2．Context Menu(上下文菜单)

当用户在某个 UI 组件（比如 ListView 的某一行）长按时弹出的菜单，有两种形式：第一种是弹出对话框式的菜单内容（见图 5-6 左），第二种是在操作栏上显示菜单（见图 5-6 右）。

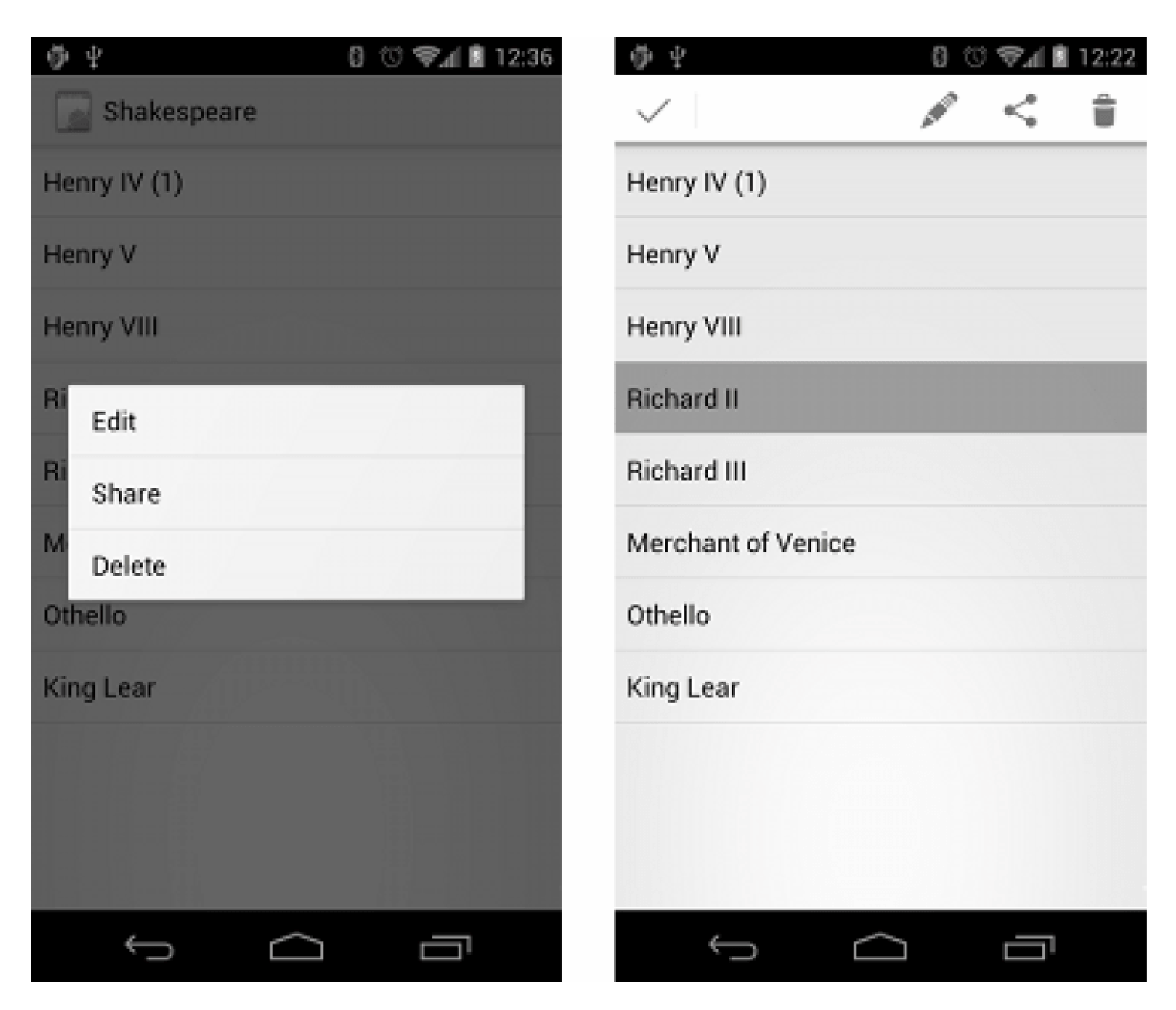


图 5-6　上下文菜单

创建上下文对话框式的菜单需要使用 registerForContextMenu(View view)将指定 UI 组件 view 与上下文菜单绑定，然后重写下列两个方法。

（1）onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo)：当该 UI 组件被长按时，系统会调用该方法以获取 Menu 选项内容。

（2）onContextItemSelected(MenuItem item)：当该菜单的某一个选项被单击时，系统会调用该方法。

为单个视图（指非 ListView 或 GridView 等有很多行的组件）创建上下文操作栏式菜单时需要实现 ActionMode.Callback 接口，在该接口回调方法中，可以指定操作栏的动作（actions）和单击每个动作时触发的事件，然后在需要显示菜单的时候调用 startActionMode()。

为 ListView 或 GridView 创建上下文操作栏式菜单时需要实现 AbsListView.MultiChoiceModeListener 接口，然后调用 setChoiceMode (CHOICE_MODE_MULTIPLE_MODAL)方法。具体步骤请参考[例 5-4]。

3．Popup Menu（弹出式菜单）

当用户单击某个按钮时，需要从多个选项中选择时，可以用 Popup Menu。Popup Menu 是锚定到 View 的模态菜单。如果空间足够，它将显示在定位视图左下方，否则显示在其左上方。它可以非常方便地在指定 view 的下面显示一个弹出菜单，类似于操作栏溢出菜单的效果。

无论是哪一种菜单，都可以使用 xml 文件创建 Menu 选项或者调用 Menu.add()通过代码动态创建所需要的 Menu 选项。

【例 5-4】 演示多种菜单的使用方法

打开 Android Studio，新建一个带空白 Activity 的项目，命名为 SpinnerDemo。打开 activity_main.xml 文件，添加三个按钮，分别用于触发上下文对话框式菜单、上下文操作栏式菜单和弹出式菜单。添加一个 ListView 用于触发上下文操作栏式菜单。ListView 的数据源使用 xml 文件绑定。

actiVity_main.xml 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout height="match parent"
    android:orientation="vertical"
    android:layout marginLeft="10dp"
    android:layout marginRight="10dp">
    <LinearLayout
        android:orientation="horizontal"
        android:layout width="match parent"
        android:layout height="wrap content"
        android:layout_marginTop="10dp">
        <Button
            android:text="上下文对话框式"
            android:layout width="0dp"
            android:layout height="match parent"
            android:id="@+id/button1"
            android:layout weight="1" />
        <Button
```

```
            android:text="上下文操作栏式"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:id="@+id/button2"
            android:layout_weight="1"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="10dp" />
        <Button
            android:text="弹出式"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:id="@+id/button3"
            android:layout_weight="1" />
    </LinearLayout>
    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/listview"
        android:entries="@array/cities"
        android:layout_marginTop="10dp" />
</LinearLayout>
```

array.xml 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="cities">
        <item>北京</item>
        <item>上海</item>
        <item>广州</item>
        <item>深圳</item>
        <item>杭州</item>
        <item>重庆</item>
        <item>成都</item>
        <item>长沙</item>
    </string-array>
</resources>
```

在 res/menu 目录（如没有此目录则新建之）下新建文件 menu.xml。输入如下：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item android:id="@+id/newfile"
        android:title="新建"
        app:showAsAction="always"
        android:icon="@android:drawable/ic_menu_add" />
    <item android:id="@+id/search"
        app:showAsAction="ifRoom"
        android:title="搜索"
        android:icon="@android:drawable/ic_menu_search" />
    <item android:id="@+id/help"
        app:showAsAction="never"
        android:title="帮助"
        android:icon="@android:drawable/ic_menu_help" />
</menu>
```

其中每个 item 标签都代表一个 menu 选项，app:showAsAction="always"表示总是以 Action 按钮方式显示；若其值为" ifRoom"，则表示如果 App Bar 有足够控件就以 Action 按钮方式显示，否则隐藏在溢出菜单中；若其值为"never "，则表示无论如何都隐藏在溢出菜单中。android:icon 属性指定菜单项的图标，但如果是弹出式菜单、上下文对话框式菜单，即便设置了图标，也是无法显示的。

打开 MainActivity.class，重载 onCreateOptionsMenu、onOptionsItemSelected、onCreateContextMenu、onContextItemSelected 四个方法，并实现 ActionMode. Callback 和 ListView. MultiChoiceModeListener 接口。

MainActivity.java 代码如下：

```
package com.example.menudemo;
import …;
public class MainActivity extends AppCompatActivity implements
 View.OnClickListener{
    Button button1, button2, button3;
    ListView listView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        button1 = (Button)findViewById(R.id.button1);
        button2 = (Button)findViewById(R.id.button2);
        button3 = (Button)findViewById(R.id.button3);
        listView = (ListView)findViewById(R.id.listview);
        registerForContextMenu(button1); //将 button1 与上下文菜单绑定
        button2.setOnClickListener(this);
        button3.setOnClickListener(this);
        menuForListView(listView); //为 listView 添加上下文菜单
    }
private ActionMode.Callback mActionModeCallback = new
 ActionMode.Callback() {
        @Override
        public boolean onCreateActionMode(ActionMode mode, Menu menu) {
            MenuInflater inflater = mode.getMenuInflater();
            inflater.inflate(R.menu.menu, menu);
            return true;
        }
        @Override
        public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
            return false;
        }
        @Override
        public boolean onActionItemClicked(ActionMode mode,
        MenuItem item) {
            switch (item.getItemId()) {
                case R.id.newfile:
                    //自定义事件处理
                    mode.finish(); // 关闭菜单
                    return true;
                case R.id.help:
                    //自定义事件处理
                    mode.finish(); // 关闭菜单
                    return true;
                case R.id.search:
```

```
                    //自定义事件处理
                    mode.finish(); // 关闭菜单
                    return true;
                default:
                    return false;
            }
        }
        @Override
        public void onDestroyActionMode(ActionMode mode) {
            mode = null;//销毁资源
        }
    };
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.menu, menu);
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.newfile:
                //自定义事件处理
                return true;
            case R.id.help:
                //自定义事件处理
                return true;
            case R.id.search:
                //自定义事件处理
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
    @Override
    public void onCreateContextMenu(ContextMenu menu, View v,
```

```
            ContextMenu.ContextMenuInfo menuInfo) {
        super.onCreateContextMenu(menu, v, menuInfo);
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.menu, menu);
    }
    @Override
    public boolean onContextItemSelected(MenuItem item) {
        AdapterView.AdapterContextMenuInfo info =
            (AdapterView.AdapterContextMenuInfo) item.getMenuInfo();
        switch (item.getItemId()) {
            case R.id.newfile:
                //自定义事件处理
                return true;
            case R.id.help:
                //自定义事件处理
                return true;
            case R.id.search:
                //自定义事件处理
                return true;
            default:
                return super.onContextItemSelected(item);
        }
    }
    private void showPopupMenu(View v){
        PopupMenu popup = new PopupMenu(this, v);
        MenuInflater inflater = popup.getMenuInflater();
        inflater.inflate(R.menu.menu, popup.getMenu());
        popup.show();
    }
    private void menuForListView(ListView listView){
        listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
        listView.setMultiChoiceModeListener(new
          AbsListView.MultiChoiceModeListener() {
            @Override
            public void onItemCheckedStateChanged(ActionMode mode, int
                position,long id, boolean checked) {
```

```
        }
        @Override
        public boolean onActionItemClicked(ActionMode mode, MenuItem
        item) {
            switch (item.getItemId()) {
                case R.id.newfile:
                    //自定义事件处理
                    mode.finish(); // 关闭菜单
                    return true;
                case R.id.help:
                    //自定义事件处理
                    mode.finish(); // 关闭菜单
                    return true;
                case R.id.search:
                    //自定义事件处理
                    mode.finish(); // 关闭菜单
                    return true;
                default:
                    return false;
            }
        }
        @Override
        public boolean onCreateActionMode(ActionMode mode, Menu
        menu) {
            MenuInflater inflater = mode.getMenuInflater();
            inflater.inflate(R.menu.menu, menu);
            return true;
        }
        @Override
        public void onDestroyActionMode(ActionMode mode) {
        }
        @Override
        public boolean onPrepareActionMode(ActionMode mode, Menu
        menu) {
            return false;
        }
```

```
153             });
154         }
155         @Override
156         public void onClick(View v) {
157             if(v == button2)
158                 startActionMode(mActionModeCallback);
159             else if(v==button3)
160                 showPopupMenu(v);
161         }
162     }
```

其中第 20～54 行实现了 ActionMode.Callback 接口，并通过第 157 行设置为在 button2 被点击时弹出上下文操作栏式菜单。第 56～60 行重载了 onCreateOptionsMenu()函数，获取了 MenuInflater 之后，将其与 menu.xml 关联，实现了选项菜单。第 62～76 行重载了 onOptionsItemSelected()函数，指定了选项菜单对应的选项被单击后触发的事件。第 78～83 行重载了 onCreateContextMenu()函数，同样也是获取了 MenuInflater 之后，将其与 menu.xml 关联，从而实现了上下文对话框式菜单。第 85～101 行重载了 onContextItemSelected()函数，指定了上下文对话框式菜单对应的选项被单击后触发的事件。第 102～107 实现了自定义函数 showPopupMenu(View)，新建了一个 PopupMenu。第 108～153 行先将 listView 的 ChoiceMode 属性设置为 ListView.CHOICE_MODE_MULTIPLE_MODAL，然后实现了它的 MultiChoiceModeListener 接口。

最后运行效果如图 5-7 和图 5-8 所示。



（a）主界面　　（b）上下文对话框式

图 5-7　Menu 样式（1）





（a）上下文操作栏式（单击按钮后）　（b）弹出式菜单　（c）上下文操作栏式（长按listView后）

图 5-8　Menu 样式（2）

5.5　Style 和 Theme

Style（样式）包含了可定义一个视图或一个窗口的外观和格式的一组属性，它将这组属性从 layout 文件中独立出来并放在<style>标签中。这些属性可以包括高度、宽度、前景色、背景色等。Style 可以重复利用，并且修改代价小，类似于网页设计的层叠样式表单（Cascading Style Sheets，CSS）。

Theme（主题）是一种特殊的 Style。一般的 Style 只是单独存储一种组件的样式，而 Theme 的样式针对整个 Activity 甚至整个应用。当一个 Activity 或应用被套用了一种 Theme 时，这个 ActiVity 或应用的所有组件对应的样式都会相同，除非重新指定。

5.5.1　使用 Style

Style 必须定义在项目中的 res/values 目录下的 xml 文件中，虽然文件名并不是强制指定的，但约定俗成的是定义在 styles.xml 文件中。该 xml 文件的根节点必须是<resources>标签。每个 Style 用一个<style>标签表示，而 Style 的每个属性都必须用<item>标签表示。item 后面紧跟关键字 name 以指明所定义的样式属性。

在不使用 Style 的情况下通常都是如下列代码所示，直接在 layout 文件中指定样式：

```
<TextView
    android:layout width="fill parent"
    android:layout height="wrap content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:text="@string/hello" />
```

如果使用 Style，则需要修改 res/values/styles.xml 文件，如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CodeFont">
        <item name="android:layout_width">fill_parent</item>
        <item name="android:layout height">wrap content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

然后在 TextView 的布局文件中指定样式即可：

```
<TextView
    style="@style/CodeFont"
    android:text="@string/hello" />
```

如此一来，大大减少了布局文件的代码数，使其结构更为清晰，有利于编程人员和设计人员的分工合作，也有利于样式的复用和统一。

5.5.2　继承 Style

<style>标签可以有两个属性，其中 name 属性用于标识 Style，必不可少，另一属性<parent>为可选属性，用于指定父 Style。因此只要将<parent>设置为另一Style，便能实现 Style 的继承。当某一组件套用了一个 Style 时，会优先使用该 Style 指定的样式属性。对于该 Style 没有指定的样式属性，则会使用父 Style 指定的。若其没有父 Style 或父 Style 也没有指定，会使用系统默认的样式属性。

一个没有指定 parent 属性的 style1 如下：

```
<style name="style1">
        <item name="android:layout_width">wrap_content</item>
        <item name="android:layout height">wrap content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
```

一个指定了以 style1 为父 Style 的 style2 如下：

```
    <style name="style2" parent="style1">
        <item name="android:layout_width">fill_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
        <item name="android:textSize">20sp </item>
    </style>
```

若有一个 TextView 如下：

```
    <style name="style2" parent="style1">
        <item name="android:layout_width">fill_parent</item>
        <item name="android:layout height">wrap content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
        <item name="android:textSize">20sp </item>
    </style>
```

那么这个 TextView 的 layout_width 属性将会是 fill_parent，字体大小是 20sp。

5.5.3　使用 Theme

鉴于 Theme 的使用对象是 Activity 乃至整个应用（application），所以一般不会直接定义一个 Theme，而是直接套用官方提供的主题或者继承官方主题后再做适当修改。官方提供了丰富多彩的主题，使用者可以通过布局文件的 design 视图下的 AppTheme 按钮中找到它们。但想让它们真正生效，必须在 mainfest 文件下的<activity>或<application>标签中指定 android:theme 属性。

Activity 套用指定主题：

```
<activity android:name=".MainActivity"
        android:theme="@android:style/Theme.Holo.Light.DarkActionBar">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
</activity>
```

应用程序套用指定主题：

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic launcher"
    android:theme="@android:style/Theme.Holo.Light.DarkActionBar">
    <activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
</activity>
</application>
```

5.5.4 继承 Theme

如前所述，Theme 是一种特殊的 Style，所以 Theme 也用<style>标签表示，因此，Theme 的继承只需要指定<parent>标签，如下面的代码所示：

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
</style>
```

然后在 mainfest 文件中指定即可：

```
<application
        android:allowBackup="true"
```

```
        android:icon="@mipmap/ic launcher"
        android:label="@string/app name"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.
                 LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
```

习 题 5

1．请完善习题 4.4 的计算器。在输入不合法的时候，使用 Toast 通知用户。

2．请完善习题 4.5 的学生信息录入系统。在输入不合法或者不完善的时候，使用 Dialog 通知用户。

3．修改上一题的学生信息录入系统，以 Spinner 的形式选择专业。

4．请思考使用 ListView 时，每次只加载部分条目，而不是全部，以提高性能。

5．请尝试为 ListView 添加上下文菜单，以支持多选删除功能。

6．Android 提供了丰富多彩的内置主题，请继承修改其中一个，以供第 3 题的应用程序使用。

第 6 章　Intent 与 Broadcast

第 4 章讲解了 Activity（活动）这一重要概念。开发者可以创建多个 Activity，但是单击应用图标后只会进入应用的主 Activity，如果想要从主 Activity 跳转到其他 Activity 就必须学习本章即将讲解的 Intent。

Intent 是 Android 开发中十分独特的一个概念。本章将讲解如何定义隐式和显式的 Intent 来启动 Activity 或 Service，如何在应用程序内和应用程序之间广播数据以及检测系统状态的变化。Broadcast Intent 可以用来在系统范围内公布应用程序事件，因此本章还将讲解 Broadcast 和 Broadcast Receiver 的相关知识。

6.1　使用 Intent 启动 Activity

Intent 作为 Android 中独特的消息传递机制，既可以在应用程序内使用，也可以在应用程序间使用。使用 Intent 来启动 Activity 的最常见方式分为显式和隐式两种：显式 Intent 使用类名显式指定要启动的类，而隐式 Intent 通过请求对一条数据执行某个动作来启动新的 Activity，这意味着开发者可以利用运行时延迟绑定要求系统启动一个可以执行给定动作的 Activity，而无须明确知道要启动的应用程序或 Activity。

此处以启动新的 Activity 为例给出相关的说明，这些相同的规则通常也适用于 Service。本书第 7 章将会详细讲述关于 Service 的相关知识。

```
Intent intent = new Intent(MyActivity.this,
MyOtherActivity.class);
startActivity(intent);
```

6.1.1　显式 Intent

Intent 有多个构造函数的重载，其中一个是 Intent(Context packageContext, Class<?> cls)。这个构造函数接收两个参数：第一个参数 Context 要求提供一个启

动 Activity 的上下文，第二个参数 Class 指定想要启动的目标活动。要显式启动一个新的 Activity，可以将构建好的 Intent 传入 Activity 类的 startActivity()方法，该方法接收一个 Intent 参数，在调用后会将新的 Activity 创建、启动和恢复运行，移动到 Activity 栈的顶部。

需要注意的是，由于新创建的 Activity 不是主 Activity，在 AndroidManifest.xml 中进行注册时无须配置<intent-filter>标签里的内容。

6.1.2 隐式 Intent

Intent 另外两个常用的构造函数分别是 Intent(String action)和 Intent(String action, Uri uri)。 这两个构造函数的第一个参数均需要指定 action 类型，第二个构造函数还可以传入 Uri 类型的对象实现程序间的数据共享。关于数据共享和 ContentProvider 的相关知识会在后续章节进行讲解。在第 5 章讲解活动在 AndroidManifest.xml 中进行注册的内容时，曾提及 activity 标签中可以添加 intent-filter 节点，以确定用来启动 Activity 的 Intent。每个 Intent Filter 可以定义一个 Activity 支持的（action）动作和多个（category）分类，只有<action>和<category>中的内容同时能够匹配上 Intent 中指定的 action 和 category 时，这个活动才能响应该 Intent。

下面的代码将 action 的字符串传入，表示使用隐式 Intent 来启动能够响应 Intent.ACTION_DIAL 这一 Android 系统内置动作的活动，此处无须指定 category，这是因为 android.intent.category.DEFAULT 作为默认的 category，在调用 startActivity()方法的时候会自动将该 category 添加到 Intent 中。当然，开发者也可以调用 Intent 中的 addCategory()方法来添加一个 category。

```
1   Intent intent = new Intent(Intent.ACTION_DIAL);
2   intent.setData(Uri.parse("tel:123-45678"));
3   startActivity(intent);
```

第二行的 setData()方法接收一个 Uri 对象，主要用于指定当前 Intent 正在操作的数据，此处通过 Uri.parse()方法将一个电话号码字符串解析成一个 Uri 对象。与此对应，开发者还可以在<intent-filter>标签中再配置一个<data>标签，用于更精确地指定当前活动能够响应什么类型的数据。<data>标签中主要可以配置以下内容。

1. android:scheme

用于指定数据的协议部分，如前面例子中的 tel 部分。

2．android:host

用于指定数据的主机名部分。

3．android:port

用于指定数据的端口部分，一般紧随在主机名之后。

4．android:path

用于指定主机名和端口之后的部分，如一段网址中跟在域名之后的内容。

5．android:mimeType

用于指定可以处理的数据类型，允许使用通配符的方式进行指定。

只有<data>标签中指定的内容和 Intent 中携带的 Data 完全一致时，当前活动才能够响应该 Intent，但是通常在<data>标签中不会指定过多的内容。

运行包含上述代码的实例，程序可以启动一个能够提供对这个电话号码进行拨号动作的新的活动，通常是手机自带的拨号程序。如果有多个活动都能够执行指定的动作，则应用程序会自动向用户呈现所有选项供用户选择。

除上面的动作外，Android 还包含大量的原生动作，表 6-1 列出了常用的 Android 原生动作。

表 6-1 常用的 Android 原生动作

动　作	说　明
ACTION_ANSWER	打开一个处理来电的 Activity
ACTION_DELETE	打开一个 Activity，删除 Intent 的 URI 中指定的数据
ACTION_DIAL	打开一个拨号程序
ACTION_EDIT	打开一个 Activity，可以编辑 Intent 的数据 URI 中的数据
ACTION_INSERT	打开一个能够在 Intent 的数据 URI 指定的游标处插入新项的 Activity，返回一个指向新插入项的 URI
ACTION_PICK	打开一个 Activity，可以从 Intent 的数据 URI 指定的 Content Provider 中选择一项，返回所选择项的 URI。例如，传递 content://contacts/people 将会调用本地联系人列表
ACTION_SEARCH	打开特定的搜索 Activity，可以使用 SearchManager.QUERY 键把搜索词作为一个 Intent 的 extra 中的字符串来提供
ACTION_WEB_SEARCH	打开浏览器，根据 SearchManager.QUERY 键提供的查询执行 Web 搜索
ACTION_SENDTO	打开一个 Activity 来向 Intent 的数据 URI 所指定的联系人发送一条消息
ACTION_SEND	打开一个 Activity，向远程联系人发送 Intent 中指定的数据，接收人需要由解析的 Activity 来选择，使用 setType 可以设置要传输的数据的 MIME 类型

续表

动　　作	说　　明
ACTION_VIEW	打开一个 Activity，以最合理的方式查看 Intent 的数据 URI 中提供的数据。一般地，http:地址会打开浏览器，tel:地址将会打开拨号程序以拨打该号码，geo:地址会在地图应用程序中显示出来，联系人信息会在联系人管理器中显示出来

如果组件指定的动作为自定义的响应动作，通常以 Java 类名和包名的完全限定名构成。与<action>标签类似，<catagory>标签也支持系统本身提供的类别和开发者自定义的类别。表 6-2 列出了常用的 Android 系统提供的类别。

表 6-2　常用的类别

值	类 别 说 明
ALTERNATIVE	默认动作的一个可替换的执行方法
SELECTED_ALTERNATIVE	与 ALTERNATIVE 类似，但替换的执行方法不是指定的，而是被解析出来的
BROWSABLE	活动可由浏览器启动
DEFAULT	为 Intent Filter 中定义的数据提供默认的动作
HOME	设备启动或按下 Home 键后显示在主屏幕的活动
LAUNCHER	应用程序启动后首先显示的活动

6.2　使用 Intent 实现 Activity 间数据传递

6.2.1　向下一个 Activity 传值

Extra 用于向 Intent 附加基本类型值，可以在任何 Intent 上使用重载后的 putExtra 方法来附加一个新的键值对，使其作为一个 Bundle 对象存储在 Intent 中。在以后启动的活动中使用对应的 getExtra 方法就可以检索它，从而实现活动间数据的传递。下面以代码为例进行详细说明。例如现在想要把一个字符串从 FirstActivity 传递到 SecondActivity，可以在启动新的活动的代码前使用 putExtra()方法。这里的 putExtra()方法接收两个参数：第一个是“键”，第二个是真正需要传递的数据。

```
String str = “Hello world”;
Intent intent = new Intent(FirstActivity.this,
SecondActivity.class);
```

```
intent.putExtra("extra data", str);
startActivity(intent);
```

然后在新开启的 Activity 中将传递的数据取出：

```
Intent intent = getIntent();
String data = intent.getStringExtra("extra data");
//使用传递过来的数据 data
```

首先通过 getIntent()方法获取到用于启动新 Activity 的 Intent，然后调用 getStringExtra()方法，传入相应的键值获得传递的数据。如果传递的数据类型为整型，则使用 getIntExtra()方法；如果传递的数据类型是布尔型，则使用 getBooleanExtra()方法，以此类推。

6.2.2 获取上一个 Activity 的返回值

当新启动的 Activity 关闭时，常常需要返回信息给先前启动的 Activity。与上一小节介绍的数据传递的不同之处在于：返回上一个 Activity 只需按手机上的返回键即可，并没有一个用于启动 Activity 的 Intent 来传递数据。因此，如果想要获取 Activity 的返回值，需要以下三个步骤：首先需要以子 Activity 的方式启动一个新的 Activity，然后在子 Activity 中设置返回值，最后在父 Activity 中获取返回值。

1．以子 Activity 的方式启动新的 Activity

使用 startActivityForResult()方法而非 startAcitvity()方法启动的 Activity 在 Activity 结束时会触发调用 Activity 内的事件处理程序 onActivityResult，从而能够在子 Activity 销毁时返回一个结果给上个 Activity。startActivityForResult()方法除了需要传入显式或隐式 Intent 来决定要启动的 Activity 以外，还需要传入一个请求码，以唯一标识返回结果的子 Activity。

```
Private static final int SHOW_SUBACTIVITY=1;
private void startSubActivity() {
Intent intent = new Intent(FirstActivity.this,
SecondActivity.class);
startActivityForResult(intent, SHOW_SUBACTIVITY);
}
```

2．设置子 Activity 返回值

子 Activity 可以在调用 finish 前调用 setResult 以向父活动返回一个结果。

setResult 方法包含两个参数：结果码和以 Intent 形式传递的结果数据。结果码表示子 Activity 的运行结果——Activity.RESULT_OK 或者 Activity.RESULT_CANCELED，还可以使用自己的响应码来处理应用程序特定的选择，setResult 支持任意的整数值。如果用户通过按下手机上的返回键关闭 Activity，或者在调用 finish 以前没有调用 setResult，那么结果码将被设为 RESULT_CANCELED，第二个参数 Intent 将被设为 null。

3．在父 Activity 中获取返回值

使用 startActivityForResult()方法启动的 Activity 在被销毁之后会回调父 Activity 内的事件处理程序 onActivityResult，因此我们需要在父 Activity 中重写这个方法来得到返回的数据，如下所示：

```
private static final int SELECT FIRST = 1;
private static final int SELECT SECOND = 2;
Uri selectedFirst = null;
Uri selectedSecond = null;
@Override
public void onActivityResult(int requestCode,int
resultCode, Intent data) {
   super.onActivityResult(requestCode,resultCode,data);
switch(requestCode)
{
      case (SELECT_FIRST):
              if (resultCode == Activity.RESULT OK)
                   selectedFirst = data.getData();
              break;
          case (SELECT SECOND):
              if (resultCode == Activity.RESULT OK)
                   selectedSecond = data.getData();
              break;
          default:
              break;
     }
}
```

onActivityResult 接收的三个参数分别为之前设置的请求码、结果码和数据 Intent。由于在一个 Activity 中有可能调用 startActivityForResult()方法启动多个不

同的 Activity，每一个 Activity 返回的数据都会回调 onActivityResult()这个方法，因此首先要通过检查请求码的值来判断数据来源。然后通过结果码的值来判断处理结果是否成功，最后从数据中取值，根据子 Activity 目标的不同，结果 Intent 可能会包含代表选定内容的 URI，也可以在返回的数据 Intent 内以 extra 的形式返回信息。

下面的实例通过一个体重计算器应用来演示 Activity 的跳转以及 Activity 间数据的传递。

【例 6-1】 BMI 标准体重计算器

打开 Android Studio，新建一个带空白 Activity 的项目，命名为 WeightCalculator。打开 activity_1.xml 文件，声明一个输入身高的 EditText、定义性别选项的 RadioGroup 和一个跳转到下一 Activity 的 Button。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:id="@+id/widget0"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android"
    >
    <TextView
        android:id="@+id/title"
        android:layout_width="wrap_content"
        android:layout_height="36dp"
        android:text="@string/title"
        android:textSize="20sp"
        />
    <TextView
        android:id="@+id/text1"
        android:layout_width="wrap_content"
        android:layout_height="96px"
        android:text="@string/text1"
        android:textSize="18sp"
        />
    <TextView
        android:id="@+id/text2"
```

```
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:text="@string/text2"
        android:textSize="18sp"
        />
    <EditText
        android:id="@+id/height"
        android:layout width="130px"
        android:layout height="wrap content"
        android:textSize="18sp"
        />

    <RadioGroup
        android:id="@+id/sex"
        android:layout width="300px"
        android:layout height="100px"
        android:layout x="97px"
        android:layout y="98px"
        android:orientation="horizontal"
        android:checkedButton="@+id/sex1"
        >

        <RadioButton
            android:id="@+id/sex1"
            android:layout_width="wrap_content"
            android:layout height="wrap content"
            android:text="男的"
            android:layout x="45dp"
            android:layout y="173dp" />
    </RadioGroup>
    <RadioButton
        android:id="@+id/sex2"
        android:layout width="wrap content"
        android:layout_height="wrap_content"
        android:text="女的"
        android:layout_x="189dp"
        android:layout_y="42dp" />

```

```
    <Button
        android:id="@+id/button1"
        android:layout_width="192px"
        android:layout_height="96px"
        android:text="计算"
        />
</LinearLayout>
```

其中，string.xml 文件内容为：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:id="@+id/widget0"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android"
    >
    <TextView
        android:id="@+id/title"
        android:layout_width="wrap_content"
        android:layout_height="36dp"
        android:text="@string/title"
        android:textSize="20sp"
        />
    <TextView
        android:id="@+id/text1"
        android:layout_width="wrap_content"
        android:layout_height="48px"
        android:text="@string/text1"
        android:textSize="18sp"
        />
    <TextView
        android:id="@+id/text2"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:text="@string/text2"
        android:textSize="18sp"
```

```
        />
    <EditText
        android:id="@+id/height"
        android:layout_width="130px"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        />

    <RadioGroup
        android:id="@+id/sex"
        android:layout_width="300px"
        android:layout_height="100px"
        android:layout_x="97px"
        android:layout_y="98px"
        android:orientation="horizontal"
        android:checkedButton="@+id/sex1"
        >

        <RadioButton
            android:id="@+id/sex1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="男的"
            android:layout_x="45dp"
            android:layout_y="173dp" />
    </RadioGroup>
    <RadioButton
        android:id="@+id/sex2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="女的"
        android:layout_x="189dp"
        android:layout_y="42dp" />

    <Button
        android:id="@+id/button1"
        android:layout_width="192px"
        android:layout_height="96px"
```

```
        android:text="计算"
        />
</LinearLayout>
```

然后在 Activity1.java 中运行 Intent 及 Bundle 对象，在调用 Activity2 时将数据传入。

```
import android.app.Activity;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.RadioButton;

public class Activity1 extends AppCompatActivity {

    private EditText et;
    private RadioButton rb1;
    private RadioButton rb2;

    @Override public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_1);

        Button b1 = (Button) findViewById(R.id.button1);
        b1.setOnClickListener(new Button.OnClickListener()
        {
            public void onClick(View v)
            {
                /*取得输入的身高*/
                et = (EditText) findViewById(R.id.height);
                double height=Double.parseDouble(et.getText().
                toString());

                /*取得选择的性别*/
```

```
                String sex=""; rb1 = (RadioButton) findViewById
                (R.id. sex1);
                rb2 = (RadioButton) findViewById(R.id.sex2);

                if(rb1.isChecked())
                {
                    sex="M";
                }else
                {
                    sex="F";
                }
                /*new 一个 Intent 对象，并指定 class*/
                Intent intent = new Intent();
                intent.setClass(Activity1.this,Activity2.class);
                /*new 一个 Bundle 对象，并将要传递的数据传入*/
                Bundle bundle = new Bundle();
                bundle.putDouble("height",height);
                bundle.putString("sex",sex);
                /*将 Bundle 对象 assign 给 Intent*/
                intent.putExtras(bundle);
                /*调用 Activity 2*/
                startActivityForResult(intent,0);
            }
        });
    }

    /* 重写 onActivityResult()*/
    @Override
    protected void onActivityResult(int requestCode,int resultCode,
    Intent data)
    {
        switch (resultCode)
        {
            case RESULT_OK:
                /* 取得数据，并显示于画面上 */
                Bundle bunde = data.getExtras();
                String sex = bunde.getString("sex");
                double height = bunde.getDouble("height");
```

```
                    et.setText(""+height);
                    if(sex.equals("M"))
                    {
                        rb1.setChecked(true);
                    }
                    else
                    {
                        rb2.setChecked(true);
                    }
                    break;
                default:
                    break;
            }
        }
    }
```

然后新建 Activity2.java 并编写 activity_2.xml 文件。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.weightcalculator.Activity2">


    <TextView
        android:id="@+id/text1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        />
    <Button
```

```
        android:id="@+id/button1"
        android:layout width="240px"
        android:layout height="96px"
        android:text="回上一页"
        />

</LinearLayout>
```

Activity2.java 文件如下：

```
import android.support.v7.app.AppCompatActivity;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.RadioButton;
import android.widget.TextView;

import java.text.DecimalFormat;
import java.text.NumberFormat;

public class Activity2 extends AppCompatActivity {

    Bundle bunde;
    Intent intent;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity 2);
        intent=this.getIntent();
        bunde = intent.getExtras();

        String sex = bunde.getString("sex");
        double height = bunde.getDouble("height");

        String sexText="";
```

```
        if(sex.equals("M"))
        {
            sexText="男性";
        }
        else
        {
            sexText="女性";
        }

        String weight=this.getWeight(sex, height);

        TextView tv1=(TextView) findViewById(R.id.text1);
        tv1.setText("你是一位"+sexText+"\n 你的身高是"+height+ "厘米
        \n 你的标准体重是"+weight+"公斤");

        Button b1 = (Button) findViewById(R.id.button1);
        b1.setOnClickListener(new Button.OnClickListener()
        {
            public void onClick(View v)
            {
                /* 回传 result 给上一个 activity */
                Activity2.this.setResult(RESULT_OK, intent);
                /* 关闭 activity */
                Activity2.this.finish();
            }
        });
    }

    /* 进行四舍五入*/
    private String format(double num)
    {
        NumberFormat formatter = new DecimalFormat("0.00");
        String s=formatter.format(num); return s; }

        private String getWeight(String sex, double height)
        {
            String weight="";
            if(sex.equals("M"))
```

```
                {
                    weight=format((height-80)*0.7);
                }
                else
                {
                    weight=format((height-70)*0.6);
                }
                return weight;
            }
}
```

运行代码，可以看到，单击“计算”按钮后，程序跳转到下一 Activity 并接收传入的数据，将计算结果显示在页面上，如图 6-1 和图 6-2 所示。

由于在 Activity1 的主程序中调用活动的方法使用的是 startActivityForResult (intent, 0)，在 Activity2 中将原本传递给其计算的 Intent 重新返回给了 Activity1，因此在此页面单击“回上一页”按钮回到上一页时，仍能保留之前输入的相关信息，如图 6-3 所示。

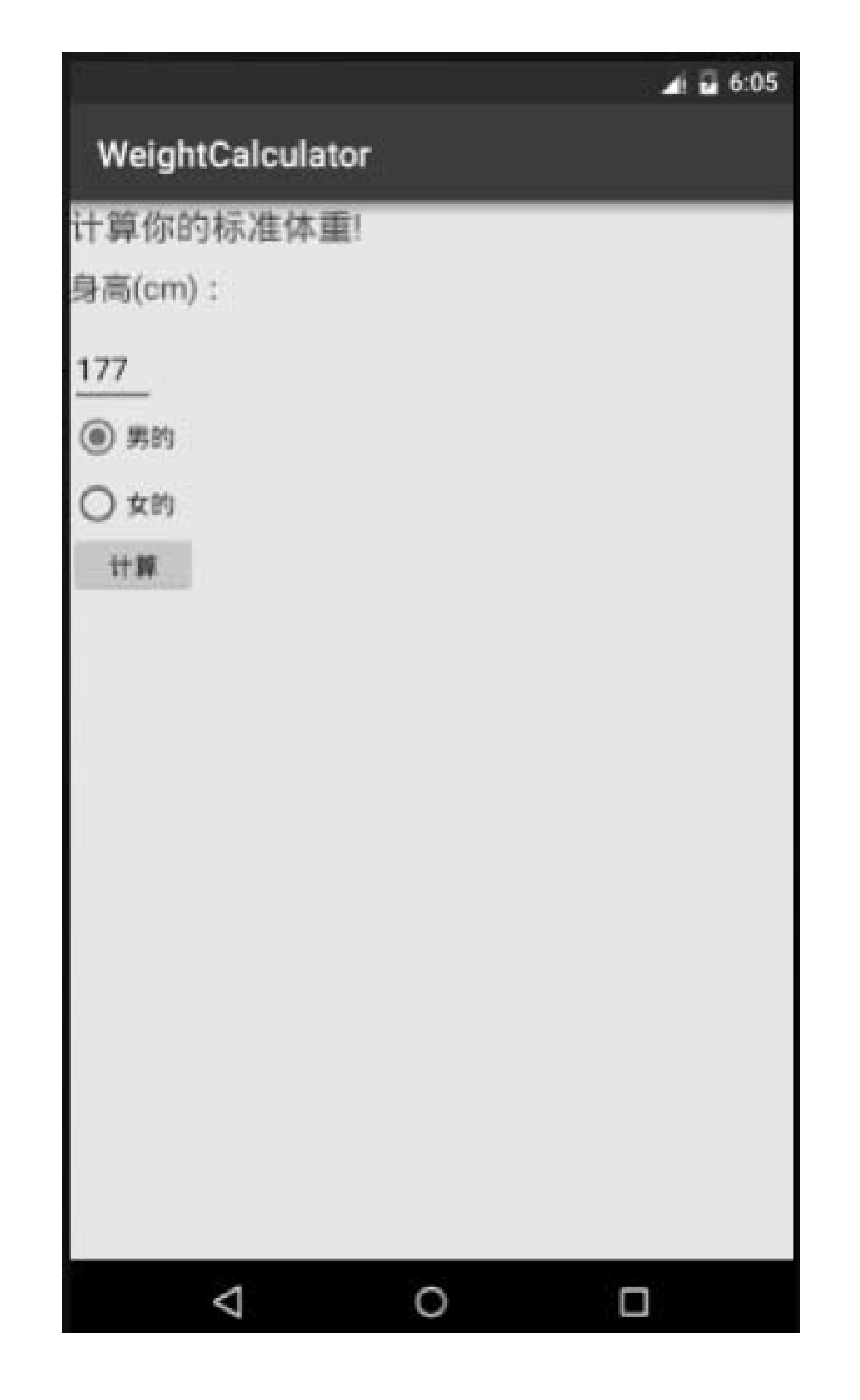


图 6-1　Activity1

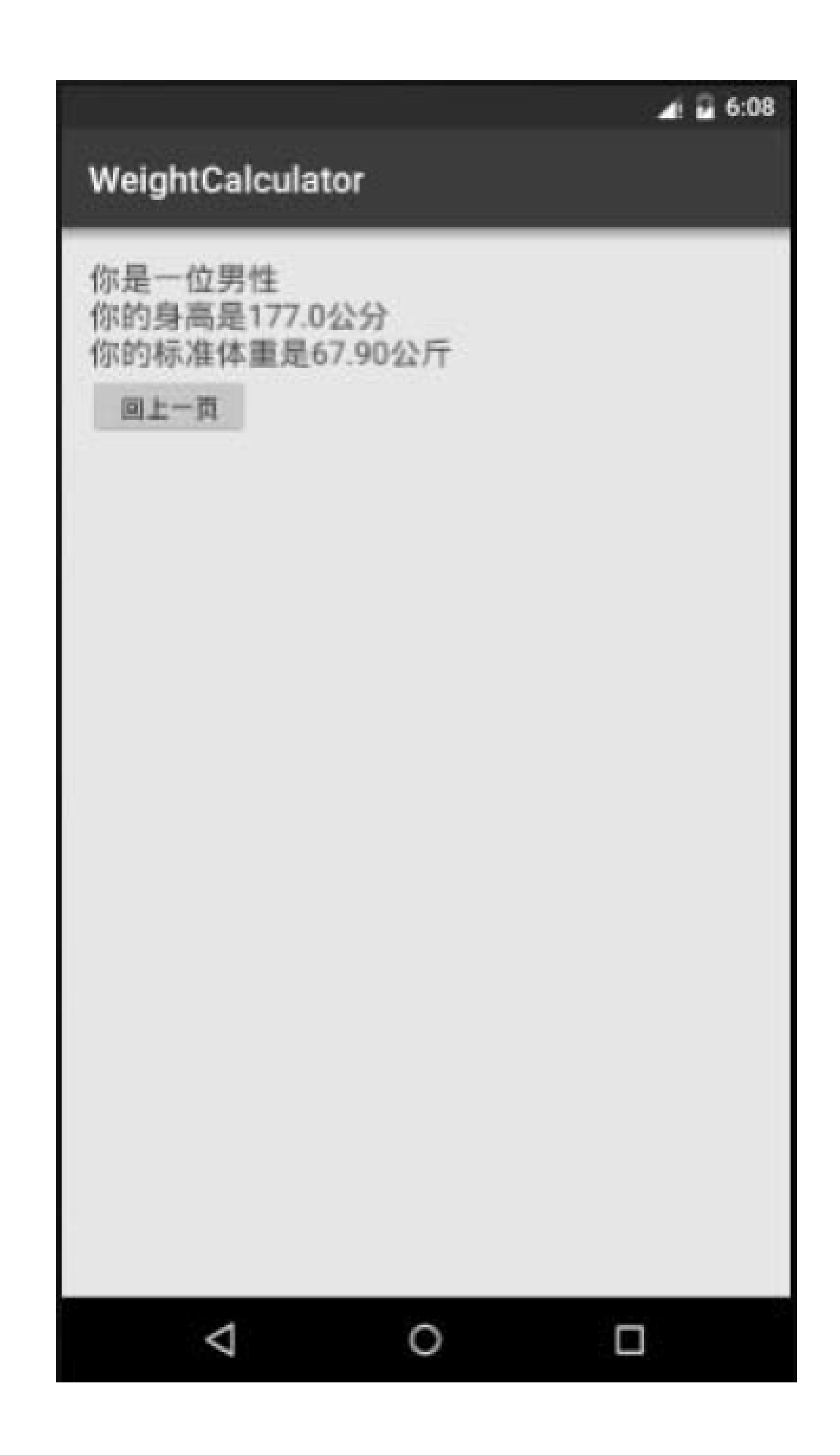


图 6-2　Activity2

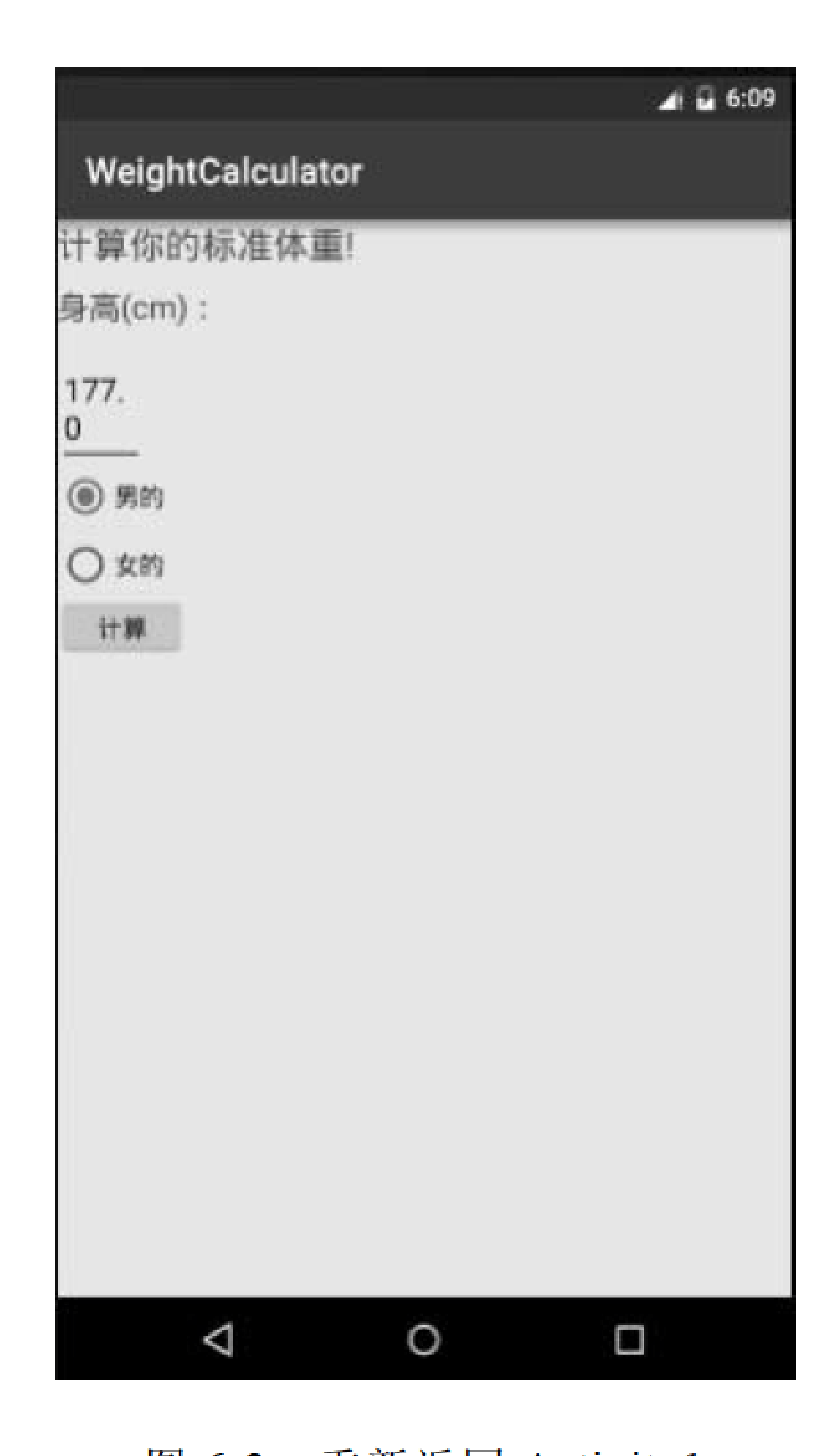


图 6-3　重新返回 Activity1

6.3　使用 Intent 广播事件

为了方便进行系统级别的消息通知，Android 中引入了广播消息机制。但相比于计算机网络中的广播，Android 中的广播机制更加灵活。因为 Android 中的每个应用程序都可以对自己感兴趣的广播进行注册，这样该程序就只会接收到自己所关心的广播内容，这些广播可能是来自于系统的，也可能是来自于其他应用程序的。Android 提供了一套完整的 API，允许应用程序自由地发送和接收广播。

Android 中的广播主要可以分为两种类型：标准广播和有序广播。

标准广播（Normal broadcast）是一种完全异步执行的广播，在广播发出之后，所有的广播接收器几乎都会在同一时刻接收到这条广播消息，因此它们之间没有任何先后顺序。这种广播的效率会比较高，但同时也意味着它是无法被截断的。

有序广播（Ordered broadcast）是一种同步执行的广播，在广播发出之后，同一时刻只会有一个广播接收器能够收到这条广播消息，当这个广播接收器中的逻辑执行完毕后，广播才会继续传递。所以此时的广播接收器是有先后顺序的，优先级高的广播接收器就可以先收到广播消息，并且前面的广播接收器还可以截断正在传递的广播，以使后面的广播接收器无法收到广播消息。

发送标准广播只需要构建一个 Intent 对象并把要发送的广播值传入，然后调用 Context 的 sendBroadcast()方法传入该 Intent 对象即可。而发送有序广播在构建 Intent 对象后要调用 sendOrderedBroadcast()方法，sendOrderedBroadcast()方法接收两个参数：第一个参数传入 Intent 实例，第二个参数是与权限相关的字符串，通常传入 null。为了体现有序广播的功能，其广播接收器也要做相应的修改：在 AndroidManifest.xml 中注册接收器的部分增加 android:priority 属性可以设置优先级，优先级较高的广播接收器可以先收到广播；在新建的广播接收器类的 onReceive()方法中调用 abortBroadcast()方法可以将广播截断从而使后面低优先级的广播接收器无法接收该条广播，实现禁止广播继续传递的功能。

系统全局广播意味着发出的广播可以被其他任何应用程序接收到，同时，开发者编写的应用程序也可以接收到来自于其他任何应用程序的广播，这样虽然实现了跨应用程序的通信但也可能带来安全隐患。为此，Android 还引入了一套本地广播机制，使其发出的广播只能在应用程序的内部进行传递且广播接收器也只能接收来自本应用程序的广播。本地广播使用 LocalBroadcastManager 来对广播进行管理，并提供了发送广播和注册广播接收器的方法。首先，通过 LocalBroadcastManager 的 getInstance()方法得到它的一个实例，然后在注册广播

接收器时调用 LocalBroadcastManager 的 registerReceiver()方法，在发送广播时调用 LocalBroadcastManager 的 sendBroadcast()方法，其余操作与系统全局广播类似。

```
private LocalBroadcastManager localBroadcastManager;
// 获取实例
localBroadcastManager
= LocalBroadcastManager.getInstance(this);
…
// 发送本地广播
localBroadcastManager.sendBroadcast(intent);
…
// 注册本地广播监听器
localBroadcastManager.registerReceiver(localReceiver,
intentFilter);
```

6.4 监 听 广 播

广播接收器可以自由地对感兴趣的广播进行注册。当有相应的广播发出时，广播接收器就能够收到该广播，并处理相应的逻辑。注册广播接收器的方式分为动态注册和静态注册两种：前者在代码中进行注册，而后者在 AndroidManifest.xml 中注册。动态注册的广播接收器可以自由地控制注册与注销，具有较强的灵活性，但是它必须要在程序启动之后才能接收到广播，因为注册的逻辑需要在 onCreate()方法中实现。当需要满足程序在未启动的情况下就能够接收到广播的要求时，必须使用静态注册的方式。

创建一个广播接收器只需新建一个类，继承 BroadcastReceiver 类并重写父类的 onReceive()方法即可。需要注意的是，在 onReceive()方法中不应该添加过多的逻辑或进行任何的耗时操作，因为在广播接收器中不允许开启线程，当 onReceive()方法运行了较长时间却没有结束，程序就会报错。因此广播接收器通常的行为是打开应用程序的其他组件。此外，Android 系统为了保证应用程序的安全性，规定：如果程序需要访问一些系统的关键性信息，必须在配置文件中声明权限，即在 AndroidManifest.xml 文件中加入<uses-permission>标签。例如下面的代码用于声明查询系统网络状态的权限：

```
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

下面的代码通过动态注册的方式编写一个能够监听网络变化的程序：

```
public class MainActivity extends Activity {
    private IntentFilter intentFilter;
    private NetworkChangeReceiver
    networkChangeReceiver;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity main);
        intentFilter = new IntentFilter();
        intentFilter.addAction("android.net.conn.
            CONNECTIVITY CHANGE");
        networkChangeReceiver = new
            NetworkChangeReceiver();
        registerReceiver(networkChangeReceiver,
            intentFilter);
     }
     @Override
     protected void onDestroy() {
         super.onDestroy();
         unregisterReceiver(networkChangeReceiver);
     }
     class NetworkChangeReceiver extends
               BroadcastReceiver
     {
     @Override
     public void onReceive(Context context, Intent
               intent) {
             //根据监测到的网络变化作相应的处理
            }
     }
}
```

首先，创建一个广播监听器：在 MainActivity 中定义了一个内部类 NetworkChangeReceiver，继承 BroadcastReceiver 并重写父类的 onReceive()方法，每当网络状态发生变化时，onReceive()方法就会得到执行。

然后在 onCreate()方法中创建一个 IntentFilter 的实例并添加一个值为 android.net.conn. CONNECTIVITY_CHANGE 的动作，表示监听网络状态的变化。接下来，创建一个 NetworkChangeReceiver 的实例，然后调用 registerReceiver() 方法进行注册，将 NetworkChangeReceiver 的实例和 IntentFilter 的实例作为参数传入，即可实现监听网络变化的功能。

注意，动态注册的广播接收器必须要取消注册，通常在 onDestroy()方法中通过调用 unregisterReceiver()方法来实现。

静态注册的广播接收器同样需要新建一个继承 BroadcastReceiver 的类，但是不再使用内部类的方式来定义，因为开发者需要在 AndroidManifest.xml 中将这个广播接收器的类名注册进去。所有静态注册的广播接收器都是通过在<application>标签内新建一个新的标签<receiver>进行注册的。它的用法其实和<activity>标签类似，首先使用 android:name 来指定具体注册的广播接收器，然后在<intent-filter>标签里声明想要接收的广播。当然，监听系统开机广播也是需要声明权限的。

习　题　6

1．请完善习题 5.6 的学生信息录入系统，使得在信息录入完，能自动跳转到第二个页面，并显示相关录入信息。

2．请设计一个可以从系统通讯录读取学生名单，并拨打电话的软件。

3．请设计一个当蓝牙状态发生改变的时候会自动广播的程序。

第 7 章　Service 与多线程

本章将讲解 Android 系统中的四大组件之一的 Service（服务），服务没有用户界面的组件，负责在后台运行执行耗时操作，并且当用户切换到另外的应用场景，它将持续在后台运行。其他应用程序组件，例如之前讲解的 Activity，可以开启 Service，其他组件也能够绑定到一个服务与之交互（IPC 机制），例如，一个 Service 可能会处理网络操作、播放音乐、操作文件 I/O 或者与 Content Provider（内容提供者）交互。

Service 并不运行在一个独立的进程中，而是依赖于创建 Service 时所在的应用程序进程。虽然 Service 在后台运行，但实际上 Service 并不会自动开启线程，所有的代码都是默认运行在主线程当中的。因此开发者需要在 Service 的内部手动创建子线程，否则就有可能出现主线程被阻塞住的情况。因此本章也会包含 Android 多线程编程的相关知识。

7.1　创建 Service

Service 需要创建一个新类继承 Service 类，并实现 Service 的 onBind()方法或者 onStartCommand()方法：

```
public class MyService extends Service {
        @Override
        public IBinder onBind(Intent intent) {
            return null;
        }
        @Override
        public void onCreate() {
            super.onCreate();
        }
        @Override
```

```
        public int onStartCommand(Intent intent, int
                    flags, int startId) {
         return super.onStartCommand(intent, flags,
                startId);
        }
        @Override
        public void onDestroy() {
            super.onDestroy();
        }
}
```

与 Activity 一样，开发者必须在 manifest 文件中声明所有的 Service。添加一个<service>元素作为<application>元素的子标签，在<service>元素中必须指定 android:name 属性，还可以包含启动 Service 的权限、Service 运行所在的进程等。

```
<manifest ... >
  …
  <application ... >
      <service android:name=".MyService" />
      …
  </application>
</manifest>
```

7.2 启动和停止服务

服务的启动有两种方式：context.startService() 和 context.bindService()，分别与服务的两种类型相对应。

1. 本地服务

本地服务（Local service）用于应用程序内部，可以调用 Context.startService()启动，调用 Context.stopService()结束。在内部可以调用 Service.stopSelf() 或 Service.stopSelfResult()自己停止，可以调用多次 startService()，但只需调用一次 stopService()停止。通过 startService()启动的服务一旦启动，服务就在后台运行，即使启动它的应用组件已被销毁。通常 started 状态的服务执行单任务并且不返回任何结果给启动者，例如从网络上下载或上传一个文件，当这项操作完成时，服务停止。

2．远程服务

远程服务（Remote Service）用于 Android 系统内部的应用程序之间，可以定义接口并把接口暴露出来，以便其他应用进行操作。客户端建立到服务对象的连接，并通过连接来调用服务，使用 Context.bindService()方法建立连接并启动，使用 Context.unbindService()关闭连接。绑定的服务只有当应用组件绑定后才能运行，多个组件可以绑定同一个服务，如果服务此时还没有加载，bindService()会先加载它，当每个组件都解绑后，该服务被销毁。

图 7-1 展示了两种启动服务方式的流程。

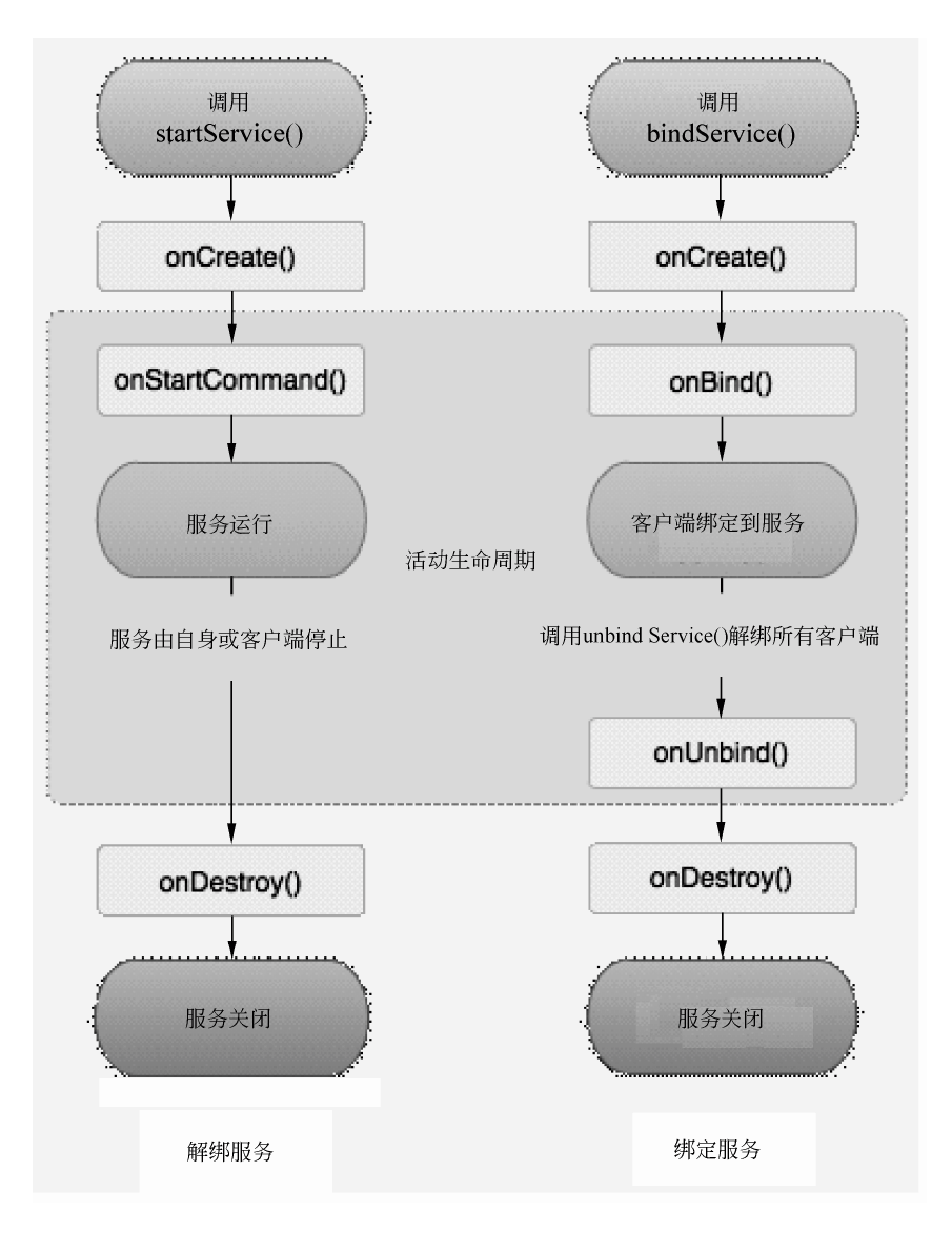


图 7-1　启动服务的两种方式的生命周期

对于 context.startService()的启动方式，如果服务还没有运行，则 Android 先调用 onCreate()，然后调用 onStart()；如果服务已经运行，则只调用 onStart()，所以一个服务的 onStartt()方法可能会被重复调用多次。如果 stopService 的时候会直接调用 onDestroy()，如果是调用组件自己直接退出而没有调用 stopService 的话，服务会一直在后台运行，该服务的调用组件再启动后可以通过 stopService 关闭 Service。所以调用 startService 的生命周期为：onCreate—>onStart（可多次调用）—>onDestroy。

对于 context.bindService()的启动方式，onBind()将返回给客户端一个 IBind 接口实例。IBind 允许客户端回调服务的方法，例如得到服务的实例、运行状态或其他操作。这时调用者会和服务绑定在一起，一旦 Context 退出服务，就会调用 onUnbind->onDestroy 相应退出。所以调用 bindService 的生命周期为：onCreate—>onBind（只可调用一次）—>onUnbind—>onDestroy。

与活动类似，服务的生命周期也涉及一些回调方法，具体如下：

```
public class ExampleService extends Service {
  int mStartMode;          //indicates how to behave
  if the service is killed
  IBinder mBinder;         //interface for clients
  that bind
  boolean mAllowRebind;    //indicates whether
  onRebind should be used

    @Override
    public void onCreate() {
        // The service is being created
    }
    @Override
    public int onStartCommand(Intent intent, int
    flags, int startId) {
        // The service is starting, due to a call to
        startService()
        return mStartMode;
    }
    @Override
    public IBinder onBind(Intent intent) {
        // A client is binding to the service with
```

```
        bindService()
        return mBinder;
    }
    @Override
    public boolean onUnbind(Intent intent) {
        // All clients have unbound with
        unbindService()
        return mAllowRebind;
    }
    @Override
    public void onRebind(Intent intent) {
        // A client is binding to the service with
        bindService(), after onUnbind() has already
        been called
    }
    @Override
    public void onDestroy() {
        // The service is no longer used and is being
        destroyed
    }
}
```

7.3 IntentService

IntentService 是 Service 类的子类，用来处理异步请求。它是一个抽象类，必须要创建子类才能使用，由于本质是服务的原因，这导致了它的优先级比单纯的线程要高很多，所以 IntentService 比较适合执行一些高优先级的后台任务。

IntentService 在处理事务时采用的是 Handler 方式，这是 Android 异步消息处理机制的一种，如果对此不熟悉，可以先跳过这部分的内容。

客户端可以通过 startService(Intent) 方法传递请求给 IntentService，IntentService 第一次启动的时候，onCreate()方法被调用：

```
@Override
public void onCreate() {
  // TODO: It would be nice to have an option to hold a partial
  wakelock
```

```
    // during processing, and to have a static startService(Context, Intent)
    // method that would launch the service & hand off a wakelock.
    super.onCreate();
    HandlerThread thread = new HandlerThread("IntentService[" + mName +"]");
    thread.start();
    mServiceLooper = thread.getLooper();
    mServiceHandler = new ServiceHandler(mServiceLooper);
}
```

onCreate()方法内部会创建一个 HandlerThread，然后使用它的 Looper 来构造一个 Handler 对象 mServiceHandler。通过 HandlerThread 单独开启一个线程来处理所有通过 startService 的方式发送过来的 Intent 请求对象所对应的任务，从而避免了事务处理阻塞主线程。执行完一个 Intent 请求对象所对应的工作之后，如果没有新的 Intent 请求达到，则自动停止 Service，否则执行下一个 Intent 请求所对应的任务。

每次启动 IntentService 的时候，它的 onStartCommand()方法就会被调用一次，IntentService 在 onStartCommand()中处理每个后台任务的 Intent，首先 onStart Command()调用 onStart()：

```
@Override
public void onStart(Intent intent, int startId) {
   Message msg = mServiceHandler.obtainMessage();
   msg.arg1 = startId;
   msg.obj = intent;
   mServiceHandler.sendMessage(msg);
}
```

这个方法通过 mServiceHandler 发送了一个消息，所以这个消息会在 HandlerThread 中被处理。消息收到后，会将 Intent 对象传递给 onHandlerIntent()方法去处理。注意这个 Intent 对象和外界 startService()参数传递的内容是一样的，通过 Intent 的参数就可以区分具体的后台任务，这样在 onHandlerIntent()方法中就可以对不同的后台任务做处理了。IntentService 的 onHandleIntent()方法是一个抽象方法，需要我们在子类中实现。需要注意的是，对于 startService()请求执行 onHandleIntent()中的耗时任务，会生成一个顺序队列，每次只有一个 Intent 传入 onHandleIntent()方法并执行。也就是说，同一时间内，只会有一个耗时任务被执

行，其他的请求必须要在后面排队，onHandleIntent()方法并不会多线程并发执行。

当 onHandlerIntent()方法执行完毕后，IntentService 会自动通过 stopSelf()方法来停止服务。不使用无参数的 stopSelf()，是因为无参函数会立刻停止服务，可能会导致还有没执行完的任务失效。有参的 stopSelf(int startId)在尝试停止服务之前会判断最近启动的服务次数是否和 startId 参数值相等，如果相等就立刻停止服务，否则反之。

IntentService 默认实现了 onBind()方法，即返回值为 null，不适合绑定的 Service。

7.4 Android 多线程编程与消息机制

由于开发者常常需要在服务内部手动创建子线程，因此本章后半部分将主要介绍 Android 多线程编程与消息机制的相关技术。应用开发需要多线程编程的场景通常基于以下三个目的：

（1）提高用户体验，防止弹出 ANR(Application is Not Responding)提示应用程序无响应对话框。

Android 应用程序的 main 线程负责处理 UI 的绘制，Android 系统为了防止应用程序反应较慢导致系统无法正常运行做了相应处理：一种情况是，当用户输入事件在 5 秒内无法得到响应时，系统会弹出 ANR 对话框，由用户决定继续等待还是强制结束应用程序；另一种情况是，当 BroadcastReciever 超过 10 秒没执行完，系统同样会弹出 ANR 对话框。因此，Android 中的 Main 线程的事件处理不能太耗时，所有可能耗时的操作都放到其他线程去处理。

（2）异步。

应用中有些情况并不一定需要同步阻塞去等待返回结果，此时可通过多线程来实现异步。例如某数据处理行为比较耗时，但可以进行异步处理，为了将 UI 与数据处理分开，需要新开一个线程来监听数据状态，包括数据状态的改变、接收数据、发送数据等等，当发现数据状态改变时通知主线程，主线程接收到通知之后进行处理，此过程还与 Android 的消息机制息息相关，因为我们需要通过消息机制发送消息到主线程并在主线程中自定义消息接口。

（3）多任务，例如多线程下载。

7.4.1 Android 多线程编程

与 Java 多线程编程相同，在 Android 多线程编程中定义一个线程只需要新

建一个类继承 Thread 类或者实现 Runnable 接口，然后重写父类的 run()方法，并在其中编写耗时逻辑即可。需要注意的是，两种方法的启动方法略有不同。

```
class MyThread extends Thread {
        @Override
        public void run() {
// 处理具体的逻辑 }
}
//继承 Thread 类对应的启动方法
new MyThread().start();

class MyThread implements Runnable {
        @Override
        public void run() {
// 处理具体的逻辑 }
}
//实现 Runnable 接口对应的启动方法
MyThread myThread = new MyThread();
new Thread(myThread).start();
```

可以看到，第二种方式中 Thread 的构造函数接收一个 Runnable 参数，而 new 的 MyThread 正是一个实现了 Runnable 接口的对象，所以可以直接将它传入到 Thread 的构造函数中，然后调用 Thread 类的 start()方法。

有关 Java 多线程的内容还有很多，例如线程优先级、线程同步等，读者可自行翻阅 Java 相关书籍进行学习。下面主要讲解 Android 中的多线程会涉及的问题，其中最主要的问题是 Android 的 UI 线程是不安全的，也就是说，在子线程中更新 UI 会抛出异常，但是有时开发者必须在子线程中执行一些耗时任务，然后根据任务的执行结果更新相应的 UI 控件。下面以代码为例讲解如何解决在子线程中进行 UI 操作的问题。

```
class MyThread extends Thread {
        @Override
        public void run() {
// 处理具体的逻辑 }
}
//继承 Thread 类对应的启动方法
new MyThread().start();
```

```

class MyThread implements Runnable {
        @Override
        public void run() {
// 处理具体的逻辑 }
}
//实现 Runnable 接口对应的启动方法
MyThread myThread = new MyThread();
new Thread(myThread).start();
```

上述代码首先定义了一个整型常量 UPDATE_TEXT 用于表示更新 TextView 这个动作，然后新建一个 Handler 对象并重写父类的 handleMessage 方法，对具体的 Message 进行处理。如果 Message 的 what 字段的值等于 UPDATE_TEXT，则将 TextView 显示的内容改成 hello world。

从关于 Change Text 按钮的单击事件中的代码中可以看到，我们并没有在子线程中直接进行 UI 操作，而是创建了一个 Message 对象并将它的 what 字段值指定为 UPDATE_TEXT，然后调用 Handler 的 sendMessage()方法将这条 Message 发送出去。很快，Handler 会收到这条 Message 并在 handleMessage()方法中对其进行处理。此时 handleMessage()方法中的代码在主线程中运行，因此是可以进行 UI 操作的。这里主要使用消息处理机制的 Handler 解决了主线程与子线程的通信问题。下面对 Android 的异步消息处理机制进行详细讲解。

7.4.2 Android 消息机制

Android 中的异步消息处理主要由四个部分组成：Message、Handler、MessageQueue 和 Looper。下面是这四个部分的简要介绍。

1. Message

Message 是在线程之间传递的消息，它可以在内部携带少量的信息，上一小节中我们使用了 Message 的 what 字段，此外还可以使用 arg1 和 arg2 字段来携带一些整型数据，使用 obj 字段携带一个 Object 对象。

生成 Message 的方法有如下三种：

（1）Message msg=new Message();

直接生成并获取消息对象。

（2）Message msg=Message.obtain();

从消息池中获取消息对象。

（3）Handler handler=new Handler();

handler.obtainMessage();

从想要发送消息的 Handler 获取消息。

2．Handler

Handler 主要用于发送和处理消息，发送消息一般使用 Handler 的 sendMessage()方法，而发出的消息经过一系列处理后，最终会传递到 Handler 的 handleMessage()方法中。

3．MessageQueue

MessageQueue（消息队列）主要用于存放所有通过 Handler 发送的消息，这部分消息会一直存在于消息队列中等待被处理，每个线程中只会有一个 MessageQueue 对象。虽然名称是队列，但是它的内部存储结构并不是真正的队列，而是采用单链表的数据结构存储消息。

4．Looper

Looper 负责管理每个线程中的 MessageQueue，若调用 Looper 的 loop()方法则 Looper 会一直尝试从 MessageQueue 中取出待处理消息并传递到 Handler 的 handleMessage()方法中。在子线程中，Looper 需要通过显式调用 Looper. Prepare()方法进行创建。Prepare 方法通过 ThreadLocal 来保证 Looper 在线程内的唯一性，如果 Looper 在线程内已经被创建并且尝试再度创建，则 Only one Looper may be created per thread 异常将被抛出。ThreadLocal 是一个线程内部的数据存储类，它可以在指定的线程中存储数据，数据存储后，只有在指定线程中可以获取到存储的数据，其他线程无法获取到数据，因此对于某些以线程为作用域的数据且不同线程具有不同的数据副本的情况，就常常采用 ThreadLocal。显然，Looper 的作用域就是线程且不同线程具有不同的 Looper，这时通过 ThreadLocal 就可以轻松实现 Looper 在线程中的存取。

Handler 在创建的时候可以指定 Looper，这样通过 Handler 的 sendMessage()方法发送出去的消息就会添加到指定 Looper 的 MessageQueue 中去。在不指定 Looper 的情况下，Handler 绑定的是创建它的线程的 Looper。如果这个线程的 Looper 不存在，那么程序将抛出 Can't create handler inside thread that has not called Looper.prepare()异常。

整个消息处理的大致流程是：

（1）包装 Message 对象——指定 Handler、回调函数和携带数据等；

（2）通过 Handler 的 sendMessage()等类似方法将 Message 发送出去；

（3）在 Handler 的处理方法里面将 Message 添加到 Handler 绑定的 Looper 的 MessageQueue；

（4）Looper 的 loop()方法通过循环不断从 MessageQueue 中提取 Message 进行处理，并移除处理完毕的 Message；

（5）通过调用 Message 绑定的 Handler 对象的 dispatchMessage()方法完成对消息的处理。在 dispatchMessage()方法中，如何处理 Message 是由用户指定的，主要包含三个判断，优先级从高到低分别是：

① Message 中的 Callback，一个实现了 Runnable 接口的对象，其中 run 函数做处理工作；

② Handler 中的 mCallback 指向一个实现了 Callback 接口的对象，由其 handleMessage 进行处理；

③ 处理消息 Handler 对象对应的类，继承并实现了其中 handleMessage 函数，通过这个实现的 handleMessage 函数处理消息。

图 7-2 展示了上述消息处理机制中四个组成部分之间的关系和整个处理过程。

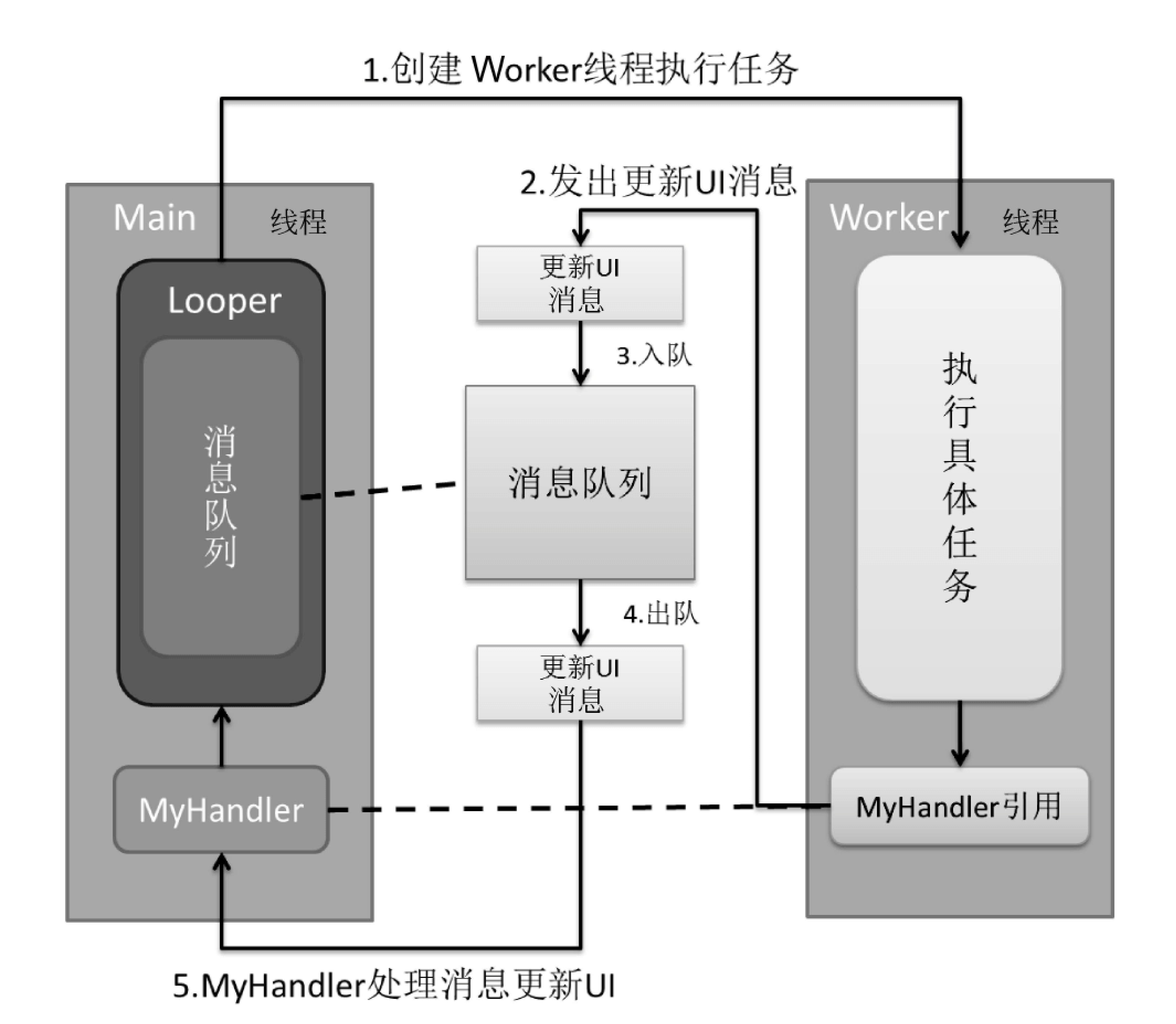


图 7-2　Android 的异步消息处理机制

从开发的角度来说，Handler 是 Android 消息机制的上层接口，这使得在开发过程中只需要与 Handler 交互。从实现的角度说，Android 的消息机制实际上就是 Handler 的运行机制，它的运行需要底层的 MessageQueue 和 Looper 的支撑。

7.4.3 使用 AsyncTask

除了上面的方法，AsyncTask 也可以用来实现线程间的通信。AsyncTask 的实现原理也是基于异步消息处理机制的，不过为了便于开发者使用，Android 对其做了很好的封装，因此，即使开发者对异步消息处理机制完全不了解也可以十分容易地实现线程间的通信。

由于 AsyncTask 是一个抽象类，所以想要使用它首先要新建一个类继承它。AsyncTask 的定义中包含三个泛型参数，分别代表“启动任务执行的输入参数”“后台任务执行的进度”“后台计算结果的类型”。在特定场合下，并不是所有类型都被使用，如果没有被使用，可以用 Java.lang.Void 类型代替。因此，一个最简单的自定义 AsyncTask 就可以写成如下方式：

```
class DownloadTask extends AsyncTask<Void, Integer, Boolean> {
…
}
```

这里把 AsyncTask 的第一个泛型参数指定为 Void，表示在执行 AsyncTask 的时候不需要传入参数给后台任务；第二个泛型参数指定为 Integer，表示使用整型数据来作为进度显示单位；第三个泛型参数指定为 Boolean，则表示使用布尔型数据来反馈执行结果。

一个异步任务的执行一般包括以下几个步骤：

（1）execute(Params... params)，执行一个异步任务，需要在代码中调用此方法，触发异步任务的执行。

（2）onPreExecute()，在 execute(Params... params)被调用后立即执行，一般用来进行一些界面上的初始化操作。

（3）doInBackground(Params... params)，在 onPreExecute()完成后立即执行，用于执行耗时的操作。此方法将接收输入参数和返回计算结果，任务一旦完成，就可以通过 return 语句来将任务的执行结果返回，当然如果 AsyncTask 的第三个泛型参数指定的是 void，就可以不返回任务执行结果。注意，在这个方法中是不可以进行 UI 操作的，如果需要更新 UI 元素，例如更新进度信息，可以在执行过

程中调用 publishProgress(Progress... values)来实现。

（4）onProgressUpdate(Progress... values)，在调用 publishProgress(Progress... values)时，此方法被执行，利用参数中的数值可以对界面元素进行相应的更新。

（5）onPostExecute(Result result)，当后台任务执行完毕并通过 return 语句进行返回时，此方法将会被调用，计算结果将作为参数传递到此方法中，直接将结果显示到 UI 组件上。

在使用的时候，有几点需要格外注意：

（1）异步任务的实例必须在 UI 线程中创建。

（2）execute(Params... params)方法必须在 UI 线程中调用。

（3）不需要手动调用 onPreExecute()、doInBackground(Params... params)、onProgressUpdate(Progress... values)、onPostExecute(Result result)这几个方法。

（4）不能在 doInBackground(Params... params)中更改 UI 组件的信息。

（5）一个任务实例只能执行一次，如果执行第二次将会抛出异常。

因此，一个比较完整的自定义 AsyncTask 就可以写成如下方式：

```
class DownloadTask extends AsyncTask<Void, Integer,
Boolean> {
    @Override
    protected void onPreExecute()
    {
        progressDialog.show(); // 显示进度对话框
    }
    @Override
    protected Boolean doInBackground(Void...
    params) {
        try {
            while (true) {
            int downloadPercent = doDownload();
            publishProgress(downloadPercent);
            if (downloadPercent >= 100) {
                break;
                }
            }
        } catch (Exception e) {
            return false;
        }
```

```
22            return true;
23        }
24        @Override
25        protected void onProgressUpdate(Integer...
26        values) {
27            // 在这里更新下载进度
28            progressDialog.setMessage("Downloaded " +
29            values[0] + "%");
30        }
31        @Override
32        protected void onPostExecute(Boolean result) {
33            // 在这里提示下载结果
34            progressDialog.dismiss(); // 关闭进度对话框
35            if (result) {
36                Toast.makeText(context, "Download
37                succeeded",Toast.LENGTH_SHORT).show();
38            } else {
39                Toast.makeText(context, " Download
40                failed",Toast.LENGTH_SHORT).show();
41            }
42        }
43    }
```

在这个 DownloadTask 中，第 9 行的 doInBackground()方法负责执行具体的下载任务，这个方法中的代码都是在子线程中运行的，因而不会影响到主线程的运行。注意在第 13 行虚构了一个 doDownload()方法，这个方法用于计算当前的下载进度并返回，此处假设这个方法已经存在了。由于 doInBackground()方法是在子线程中运行的，所以肯定不能进行 UI 操作，但可以调用 publishProgress()方法并将当前的下载进度传进来，这样 onProgressUpdate()方法就会很快被调用，并在这里进行 UI 操作。

当下载完成后，doInBackground()方法会返回一个布尔型变量，这样 onPostExecute()方法就会很快被调用，这个方法也是在主线程中运行的。然后根据下载的结果弹出相应的 Toast 提示，从而完成整个 DownloadTask 任务。如果想要启动这个任务，只需编写以下代码即可：

```
new DownloadTask().execute();
```

通过本实例可以看到，AsyncTask 很好地对异步消息处理机制进行了封装，开发

者无须专门使用一个 Handler 来发送和接收消息，只需调用 publishProgress()方法就可以轻松地从子线程切换到 UI 线程。 理解 Android 应用程序消息处理模型有利于我们在开发 Android 应用程序时，充分利用多线程的并发性来提高应用程序的性能，获得良好的用户体验。

7.4.4 线程池

在操作系统中，线程是 CPU 调度的最小单位，同时线程又是一种受限的资源，其创建和销毁都会有相应的开销，因此不可能无限制地产生。在一个进程中频繁地创建和销毁线程显然不是高效的做法，更好的方法是使用线程池，在一个线程池中缓存一定数量的线程，从而避免因为频繁创建和销毁线程带来的系统开销。Android 中的线程池来源于 Java，主要是通过 Executor 来派生特定类型的线程池，不同类型的线程池具有各自的特性。

Executor 本身是一个接口，ThreadPoolExecutor 是该接口的一个实现类，是 Android 中线程池的真正实现。ThreadPoolExecutor 的构造函数提供了一系列参数来配置线程池。

```
public ThreadPoolExecutor(int corePoolSize,
                          int maximumPoolSize,
                          long keepAliveTime,
                          TimeUnit unit,
                            BlockingQueue<Runnable>
                            workQueue,
                          ThreadFactory threadFactory,
                            RejectedExecutionHandler
                            handler)
```

下面对这个构造函数的七个参数的含义进行介绍。

corePoolSize：线程池中核心线程的数量，默认情况下，核心线程会一直在线程池中存活，即使它们处于闲置状态。

maximumPoolSize：线程池中最大线程数量，线程池中线程数量达到该数值后，后续的新任务会被阻塞。

keepAliveTime：非核心线程的超时时长，当系统中非核心线程的闲置时间超过 keepAliveTime 这个时长之后就会被回收。如果将 ThreadPoolExecutor 的 allowCoreThreadTimeOut 属性设置为 true，则该参数也表示核心线程的超时时长。

unit：第三个参数的单位，有纳秒、微秒、毫秒、秒、分、时、天等。

workQueue：线程池中的任务队列，该队列主要用来存储已经被提交但是尚未执行的任务。存储在这里的任务是由 ThreadPoolExecutor 的 execute 方法提交来的。workQueue 是 BlockingQueue 类型的，这是一个特殊的队列，当我们从 BlockingQueue 中取数据时，如果 BlockingQueue 是空的，则取数据的操作会进入到阻塞状态，当 BlockingQueue 中有新数据时，这个取数据的操作又会被重新唤醒。同理，如果 BlockingQueue 中的数据已满，那么向 BlockingQueue 中存数据的操作会进入阻塞状态，直到 BlockingQueue 中有新的空间。

（1）ArrayBlockingQueue：表示一个规定了大小的 BlockingQueue，Array BlockingQueue 的构造函数接受一个 int 类型的数据，该数据表示 BlockingQueue 的大小，存储在 ArrayBlockingQueue 中的元素按照 FIFO（先进先出）的方式来进行存取。

（2）LinkedBlockingQueue：表示一个大小不确定的 BlockingQueue，在 Linked BlockingQueue 的构造方法中可以传一个 int 类型的数据，这样创建出来的 LinkedBlockingQueue 是有大小的，也可以不传，不传的话，LinkedBlockingQueue 的大小就为 Integer.MAX_VALUE。

threadFactory：为线程池提供创建新线程的功能，一般使用默认设置即可。threadFactory 接口只有方法：Thread newThread(Runnable r)。

handler 拒绝策略：当线程无法执行新任务时（一般是由于线程池中的线程数量已经达到最大数或者线程池关闭导致的）执行的策略。默认情况下，当线程池无法处理新线程时，会抛出一个 RejectedExecutionException。

线程池执行任务时大致遵循下列运行规则：

（1）如果线程池中的数量为达到核心线程的数量，则直接会启动一个核心线程来执行任务。

（2）如果线程池中的数量已经达到或超过核心线程的数量，则任务会被插入到任务队列中等待执行。

（3）如果(2)中的任务无法插入到任务队列中，由于任务队列已满，这时如果线程数量未达到线程池规定最大值，则会启动一个非核心线程来执行任务。

（4）如果(3)中线程数量已经达到线程池最大值，则会拒绝执行此任务，ThreadPoolExecutor 会调用 RejectedExecutionHandler 的 rejectedExecution 方法通知调用者。

接下来，对 Android 中最常见的四类具有不同功能特性的线程池进行介绍。它们都直接或间接地通过配置上述的 ThreadPoolExecutor 来实现自己的功能特性。

（1）FixedThreadPool

该模式全部由核心线程去实现，并不会被回收，没有超时限制和任务队列的限制，会创建一个固定线程池，可控制线程最大并发数，超出的线程会在队列中等待。实现代码如下：

```
public static ExecutorService newFixedThreadPool(int
mThreads){
    return new ThreadPoolExecutor(mThreads,mThreads,
                                  0L,TimeUtil.MILL
                                  ISECONDS,new
                                  LinkedBlockingQueue());
}
```

（2）CachedThreadPool

该模式下线程数量不固定，只有非核心线程，最大值为 Integer.MAX_VALUE，如果线程池长度超过处理需要，可灵活回收空闲线程；若无可回收，则新建线程。实现代码如下：

```
public static ExecutorService newCachedThreadPool(){
    return new ThreadPoolExecutor(0,Integer.MAX VALUE,
                            60L,TimeUtil.SECONDS,new
                                SynchronousQueue());
}
```

（3）ScheduledThreadPool

该模式下核心线程是固定的，非核心线程没有限制，非核心线程闲置时会被回收，负责执行定时任务和固定周期的任务。实现代码如下：

```
public static ScheduledExecutorService
newScheduledThreadPool(int corePoolSize){
          return new
          SchduledThreadPoolExecutor(corePoolSize);
}

public SchduledThreadPoolExecutor(int corePoolSize){
     super(corePoolSize,Integer.MAX VALUE,0,
            NANOSECONDS,new DelayedWorkQueue());
}
```

（4）newSingleThreadExecutor

该模式下线程池内部只有一个线程，所有的任务都在一个线程中执行，会创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序（FIFO、LIFO、优先级）执行。实现代码如下：

```
public static ExecutorService
newSingleThreadExecutor(){
        return new
                 FinalizableDelegatedExecutorService
                (newThreadPoolExecutor
                     (1,1,0L,TimeUtil.MILLISECONDS,
                       new LinkedBlockingQueue()));
}
```

下面的示例展示了四种模式的线程池的具体使用方法：

```
Runnable runnable = new Runnable(){
        public void run(){
                SystemClock.sleep(2000);
         }
}

ExecutorService fixedThreadPool =
Executors.newFixedThreadPool(4);
fixedThreadPool.execute(runnable);

ExecutorService cachedThreadPool =
Executors.newCachedThreadPool();
cachedThreadPool.execute(runnable);

ScheduledExecutorService scheduledThereadPool =
Executors.newScheduledThreadPool(4);
scheduledThereadPool.schedule()runnable,2000,TimeUti
l.MAX_VALUE);//2000ms 后执行。
//延迟 10ms 后，每隔 1000ms 执行一次
scheduledThereadPool.scheduleAtFixedRate(runnable,10
,1000,TimeUtil.MILLISECONDS);
```

```
ExecutorService  singleThreadExecutor =
Executors.newSingleThreadExecutor();
singleThreadExecutor.execute(runnable);
```

最后，对 Android 中的线程形态做一个全面的总结，除了 Thread 本身，Android 中能够扮演线程角色的还包含 AsyncTask、IntentService 和 HandlerThread。虽然它们的表现形式各有差别，但它们的本质仍然是传统的线程。AsyncTask 封装了线程池和 Handler，方便开发者在子线程中更新 UI。HandlerThread 是一种具有消息循环的线程，在它的内部可以使用 Handler。IntentService 内部采用 HandlerThread 来执行任务，执行完毕后 IntentService 会自动退出，使得开发者能够更方便地执行后台任务。

习　题　7

1. 请设计一个简易的播放器，其应该支持后台播放。
2. 如何在子线程中修改 UI?
3. Android 如何实现线程间的通信？

第 8 章　数据持久化技术和 ContentProvider

本章主要讲解 Android 系统的数据持久化方法。应用程序在执行过程中常常需要长久存储某些数据以便以后取出并应用，Android 系统提供了多种数据存储的方式，包括 XML 文件的 SharedPreferences 存储方式、文件存储方式和 SQLite 数据库存储方式。然而，使用这些持久化技术所保存的数据都只能在当前应用程序中访问，为了实现跨程序数据共享的功能，本章还将讲解更加安全可靠的 Content Provider（内容提供器）技术。

8.1　SharedPreference

SharedPreferences 类是一个轻量级的存储类，特别适合用于保存软件配置参数，例如用户个性化设置的字体、颜色、位置等参数信息。一般的应用程序都会提供“设置”或者“首选项”这样的选项，这些设置最后就可以通过 SharedPreferences 来保存。它为开发者提供了一个可以通过 key-value（键值对）来存取数据的功能，但数据仅限于基本数据类型和字符串。使用 SharedPreferences 保存数据，其背后是用 xml 文件存放数据，文件存放在/data/data/<package name>/shared_prefs 目录下。

8.1.1　获取 SharedPreferences 对象方法

通过调用以下的方法之一可以创建新的 SharedPreferences 文件或访问现有的文件：

```
SharedPreferences pre =
Context.getSharedPreferences(String name,int mode);
```

其中，第一个参数 name 为组件的配置文件名，若要按照第一个参数指定的名称识别的多个共享首选项文件，可以使用此方法。注意，因为在 Android 中已

经确定了 SharedPreferences 是以 xml 形式保存的，所以，在填写文件名参数时无须给定.xml 后缀，Android 会自动添加并将其直接保存在/data/data/<package name>/shared_prefs 目录下。

```
SharedPreferences pre = Activity.getPreferences(int
mode);
```

配置文件仅可以被调用的 Activity 使用，如果只需使用 Activity 的一个共享首选项，则可以使用该方法。其中 mode 为操作模式，默认的模式为 0 或 MODE_PRIVATE，还可以使用 MODE_APPEND、MODE_WORLD_READABLE 和 MODE_WORLD_WRITEABLE。

除了以上两种方法，每个应用都有一个默认的配置文件 preferences.xml，可以使用 getDefaultSharedPreferences 方法获取。

8.1.2 写入 SharedPreferences

要写入 SharedPreferences，可以通过对 SharedPreferences 调用 edit()来创建一个 SharedPreferences.Editor，并使用 putInt()和 putString()方法传递写入的键和值；然后，调用 commit()以保存所做的更改：

```
SharedPreferences sharedPref =
getActivity().getPreferences(Context.MODE PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putInt(getString(R.string.saved high score),
newHighScore);
editor.commit();
```

8.1.3 从 SharedPreferences 读取信息

要从 SharedPreferences 中检索值，请调用 getInt()和 getString()等方法，并为想要的值提供键，还可以根据需要提供键不存在的情况下返回的默认值：

```
SharedPreferences sharedPref =
getActivity().getPreferences(Context.MODE PRIVATE);
int defaultValue =
getResources().getInteger(R.string.saved_high_score_
default);
```

```
long highScore =
sharedPref.getInt(getString(R.string.saved high score
), defaultValue);
```

8.2 文　件

Android 使用与其他平台上基于磁盘的文件系统类似的文件系统。本小结讲述如何使用 Android 文件系统读取和写入文件。本书假设你已经掌握 Java 负责文件和目录管理的 File 类以及 IO 文件流的相关知识，如果还不熟悉相关内容的，可以翻阅相关 Java 教材进行复习。所有 Android 设备都有两个文件存储区域："内部"存储和"外部"存储。之所以存在这两种分类，是因为大多数手机设备都提供内置的非易失性内存（即内部存储）和移动存储介质（例如微型 SD 卡，即外部存储)。如今，即便设备没有移动存储介质，也会将永久性存储空间划分为"内部"和"外部"分区，从而保证始终有两个存储空间。当然，无论外部存储是否可移动，API 的使用方法并不受影响。

8.2.1 内部存储

内部存储位于系统中很特殊的一个位置，如果将文件存储于内部存储空间中，那么文件默认只能被本应用访问到，且一个应用所创建的所有文件都在和应用包名相同的目录下，当一个应用卸载之后，内部存储中的这些文件也被删除。SharedPreferences 和 SQLite 数据库都是存储在内部存储空间上的，需要注意的是，内部存储空间十分有限，因而十分可贵，另外，它也是系统本身和系统应用程序主要的数据存储所在地，一旦内部存储空间耗尽，手机就无法正常使用了，所以对于内部存储空间，要尽量避免使用。

内部存储一般用 Context 来获取和操作。使用 getFilesDir()方法可以获取 App 的内部存储空间，准确地说，是应用在内部存储上的根目录。下面列举一些内部存储的常见操作：

```
File file = newFile(context.getFilesDir(), filename);
```

（1）创建内部存储文件。

（2）读写内部存储空间上的文件。

读取内部存储的文件，可以使用 Context.openFileInput() 打开指定名称的文

件，并返回 FileInputStream 对象供数据读取之用。

```
String filename = "myfile";
String string = "Hello world!";
FileOutputStream outputStream;
try{
    outputStream = openFileOutput(filename,
                       Context.MODE_PRIVATE);
    outputStream.write(string.getBytes());
    outputStream.close();
} catch(Exception e) {
    e.printStackTrace();
}
```

（3）列出所有已创建的文件。

```
String[] files = Context.fileList();
for(String file : files)  {
    Log.e(TAG, "file is "+ file);
}
```

（4）删除文件。

```
if(Context.deleteFile(filename)) {
Log.e(TAG, "delete file "+ filename + "
sucessfully“);
} else {
    Log.e(TAG, "failed to deletefile " + filename);
}
```

（5）创建目录。

```
File workDir = Context.getDir(dirName,
Context.MODE PRIVATE);
Log.e(TAG, "workdir "+ workDir.getAbsolutePath();
```

8.2.2 外部存储

外部存储中的文件是可以被用户或者其他应用程序修改的，主要有两种类型的文件（目录）：公有文件和私有文件。由于是外部存储，两种文件类型都是可以被其他应用程序访问的，虽然对私有文件的访问对非恶意程序而言是没有意义的。两种文件类型的区别主要在于当应用被卸载之后，对于公有文件，其卸载前创建的文件仍然保留，而私有文件会被删除。

要对外部存储进行文件操作，必须在清单文件中先加上权限设置。

1）可写：

```
<manifest …>
<uses-permission
android:name="android.permission.WRITE EXTERNAL STORAGE" />
    …
</manifest>
```

2）可读：

```
<manifest …>
<uses-permission
android:name="android.permission. READ_EXTERNAL_STORAGE" />
    …
</manifest>
```

3）创建及修改：

```
<manifest …>
 <uses-permission
android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
    …
</manifest>
```

创建外部存储文件时，一般建议指定要存储的目录类型，这样系统会自动创建对应的目录。这些目录类型是定义在 Environment 类中的字符串常数，共有下列几种：

DIRECTORY_ALARMS——系统提醒铃声存放的标准目录。

DIRECTORY_DCIM——相机拍摄照片和视频的标准目录。

DIRECTORY_DOWNLOADS——下载的标准目录。

DIRECTORY_MOVIES——电影存放的标准目录。

DIRECTORY_MUSIC——音乐存放的标准目录。

DIRECTORY_NOTIFICATIONS——系统通知铃声存放的标准目录。

DIRECTORY_PICTURES——图片存放的标准目录。

DIRECTORY_PODCASTS——系统广播存放的标准目录。

DIRECTORY_RINGTONES——系统铃声存放的标准目录。

当然，外部存储并不总是可用的，例如用户可能已将存储装载到计算机中或已移除了提供外部存储的 SD 卡。因此，在访问它之前，要先调用 getExternalStorageState() 查询外部存储状态，如果返回的状态为 MEDIA_MOUNTED，则可以对文件进行读写。

```
//检查外部存储是否可读写
public boolean isExternalStorageWritable() {
    String state =
    Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}
//检查外部存储是否可读
public boolean isExternalStorageReadable() {
    String state =
    Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        return true;
    }
    return false;
}
```

在外部存储中，创建应用私有文件的方法是 getExternalFilesDir()，创建公有文件的方法是 getExternalStoragePublicDirectory()。API 8 以下的版本在操作文件的时候没有专门区分并分别为私有文件和公共文件的操作提供 API 支持，开发者

只能先使用 Environment.getExternalStorageDirectory()获取根目录再自行进行相应的操作。

```
//创建公有 albumName 文件
public File getAlbumStorageDir(String albumName) {
     File file =
     newFile(Environment.getExternalStoragePublicDire
     ctory(Environment.DIRECTORY PICTURES),
     albumName);
     if(!file.mkdirs()) {
        Log.e(LOG_TAG, "Directory not created");
    }
    return file;
}
//创建私有 albumName 文件
public File getAlbumStorageDir(Context context,
String albumName) {
    File file = newFile(context.getExternalFilesDir(
    Environment.DIRECTORY PICTURES), albumName);
    if(!file.mkdirs()) {
        Log.e(LOG TAG, "Directory not created");
    }
    return file;
}
```

所有应用程序的外部存储的私有文件都放在根目录的 Android/data/下，目录形式为/Android/data/<package_name>/。需要注意的是，对于存放在数据区(/data/data/..)的外部存储文件进行读写操作需要先调用 openFileOutput 和 openFileInput 方法，而不能直接使用 FileInputStream 和 FileOutputStream 进行文件的操作。

1）写操作：

```
FileOutputStream fout =openFileOutput(fileName, MODE_PRIVATE);
```

2）读操作：

```
FileInputStream fin = openFileInput(fileName);
```

3）写操作中的使用模式：

MODE_APPEND——向文件尾写入数据。

MODE_PRIVATE——仅打开文件可写入数据。

MODE_WORLD_READABLE——所有程序均可读该文件数据。

MODE_WORLD_WRITABLE——即所有程序均可写入数据。

【例 8-1】 读写/data/data/<应用程序名>目录下的文件

```
//写数据
 public void writeFile(String fileName,String
writestr) throws IOException{
  try{

FileOutputStream fout =openFileOutput(fileName,
MODE PRIVATE);

    byte [] bytes = writestr.getBytes();

    fout.write(bytes);

    fout.close();
  }

  catch(Exception e){
        e.printStackTrace();
  }
 }

//读数据
public String readFile(String fileName) throws
IOException{
    String res="";
    try{
         FileInputStream fin =
                   openFileInput(fileName);
         int length = fin.available();
         byte [] buffer = new byte[length];
         fin.read(buffer);
         res = EncodingUtils.getString(buffer, "UTF-8");
```

```
            fin.close();
        }
        catch(Exception e){
            e.printStackTrace();
    }
    return res;
    }
```

对于外部存储 sdcard 的数据，直接使用 Java 文件流 FileInputStream 和 FileOutputStream 进行相关读写操作即可。

【例 8-2】 读写/mnt/sdcard/目录下的文件

```
//写数据到/mnt/sdcard 中的文件
public void writeFileSdcardFile(String
fileName,String write str) throws IOException{
 try{
   FileOutputStream fout = new
   FileOutputStream(fileName);
   byte [] bytes = write_str.getBytes();
   fout.write(bytes);
   fout.close();
 }
catch(Exception e){
    e.printStackTrace();
   }
 }
//读/mnt/sdcard 中的文件
public String readFileSdcardFile(String fileName)
throws IOException{
  String res="";
  try{
     FileInputStream fin = new
     FileInputStream(fileName);
```

```

        int length = fin.available();

        byte [] buffer = new byte[length];
        fin.read(buffer);

        res = EncodingUtils.getString(buffer, "UTF-8");

        fin.close();
       }

       catch(Exception e){
           e.printStackTrace();
       }
return res;
}
```

8.2.3 资源文件的读取

应用程序常用的资源一般都会置于项目的 res 目录中，但有些资源不适合放在 res 目录中管理，例如一篇几万字的小说，这种资源建议放在 assets 目录下。在 assets 目录中的文件不能通过资源 ID 来获取，必须通过较低级的 I/O 方式来获取，且这些数据只能读取，不能写入，更重要的是，该目录下的文件大小不能超过 1MB。

【例 8-3】 从 resource 的 asset 中读取文件数据

```
String fileName = "test.txt";
 String res="";
  try{

  //得到资源中的 asset 数据流
  InputStream in =
  getResources().getAssets().open(fileName);

  int length = in.available();
  byte [] buffer = new byte[length];

```

```
    in.read(buffer);
    in.close();
    res = EncodingUtils.getString(buffer, "UTF-8");

    }catch(Exception e){

    e.printStackTrace();
  }
```

【例 8-4】 从 resource 的 raw 中读取文件数据

```
String res = "";
try{

//得到资源中的 Raw 数据流
InputStream in =
getResources().openRawResource(R.raw.test);

//得到数据的大小
int length = in.available();

byte [] buffer = new byte[length];

//读取数据
in.read(buffer);

//依 test.txt 的编码类型选择合适的编码，如果不调整会出现乱码
res = EncodingUtils.getString(buffer, "BIG5");

//关闭
in.close();

 }catch(Exception e){
  e.printStackTrace();
 }
```

apk 资源文件的大小不能超过 1MB，如果超过的话，可以将这个数据先复制

到 data 目录下，然后再使用。

【例 8-5】 复制数据至 data 目录

```
public boolean assetsCopyData(String
    strAssetsFilePath, String strDesFilePath){
      boolean bIsSuc = true;
      InputStream inputStream = null;
      OutputStream outputStream = null;

      File file = new File(strDesFilePath);
      if (!file.exists()){
          try {
             file.createNewFile();
             Runtime.getRuntime().exec("chmod 766 "
                   + file);
          } catch (IOException e) {
             bIsSuc = false;
          }
      }else{
          return true;
      }

      try {
          inputStream =
                getAssets().open(strAssetsFilePath);
          outputStream = new
                FileOutputStream(file);
          int nLen = 0 ;
          byte[] buff = new byte[1024*1];
          while((nLen=inputStream.read(buff))> 0){
             outputStream.write(buff, 0, nLen);
          }
      } catch (IOException e) {
          bIsSuc = false;
      }finally{
          try {
             if (outputStream != null){
                 outputStream.close();
```

```
                }

                if (inputStream != null){
                    inputStream.close();
                }
            } catch (IOException e) {
                bIsSuc = false;
            }

        }

        return bIsSuc;
    }
```

8.3 SQLite

将数据保存到数据库对于重复或结构化数据而言是理想之选，本节假设你基本熟悉 SQL 数据库并讲解 Android 中 SQLite 数据库的使用。SQLite 数据库是一个开源的嵌入式 SQL 数据库引擎，且属于关系型数据库，它与其他 Server 等级的数据库最大的差异在于：SQLite 直接将数据库的数据存储在本机端，而非 Server 端，而且一个数据库即存储成一个文件，非常简单且数据库系统十分轻量，仅为几百 KB 大小。

Android 运行时环境包含了完整的 SQLite，数据库存储在 data/<项目文件夹>/databases/下。SQLite 对数据类型的支持与其他主要的 SQL 数据库稍有不同，创建一个表时在 CREATE TABLE 语句中可以指定某列的数据类型，但是之后仍然可以把任何数据类型放入任何列中。当某个值插入数据库时，SQLite 将检查它的类型，如果该类型与关联的列不匹配，则 SQLite 会尝试将该值转换成该列的类型；如果不能转换，则该值将作为其本身具有的类型存储。例如可以把一个字符串（String）放入 INTEGER 列，SQLite 称其为“弱类型”（manifest typing）。

此外，SQLite 不支持一些标准的 SQL 功能，例如外键约束、嵌套 transaction、RIGHT OUTER JOIN 和 FULL OUTER JOIN，还有一些 ALTER TABLE 功能。除上述功能外，SQLite 是一个完整的 SQL 系统，拥有完整的触发器、交易等。

活动可以通过 Content Provider 或者 Service 访问一个数据库，下面将详细讲

解如何创建数据库、添加数据和查询数据库。

8.3.1 数据库创建

Android 不自动提供数据库，在 Android 应用程序中使用 SQLite，必须自己创建数据库，然后创建表、索引，填充数据。Android 提供了 SQLiteOpenHelper 类用于创建一个数据库，开发时只要继承 SQLiteOpenHelper 类就可以轻松地创建数据库。SQLiteOpenHelper 类根据开发应用程序的需要，封装了创建和更新数据库使用的逻辑。SQLiteOpenHelper 的子类至少需要实现三个方法：

（1）构造函数，即调用父类 SQLiteOpenHelper 的构造函数，这个方法需要四个参数：上下文环境、数据库名字、可选的游标工厂（通常是 Null）和一个代表正在使用的数据库模型版本的整数。

（2）onCreate()方法，它需要一个 SQLiteDatabase 对象作为参数，根据需要对这个对象填充表和初始化数据。下面的示例在建表时调用了通用的 execSQL(String sql)方法执行 SQL 语句。

（3）onUpgrade()方法，它需要三个参数：一个 SQLiteDatabase 对象、一个旧的版本号和一个新的版本号，这样可以把一个数据库从旧的模型转变到新的模型。

【例 8-6】 创建订单数据库

```
public class OrderDBHelper extends SQLiteOpenHelper{
    private static final int DB_VERSION = 1;
    private static final String DB_NAME = "myTest.db";
    public static final String TABLE_NAME = "Orders";

    public OrderDBHelper(Context context) {
        super(context, DB_NAME, null, DB_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase
    sqLiteDatabase) {
        // create table Orders(
        String sql = "create table if not exists " +
                        TABLE_NAME + " (Id integer
                        primary key, CustomName text,
                        OrderPrice integer, Country
                        text)";
```

```
        sqLiteDatabase.execSQL(sql);
    }

    @Override
    public void onUpgrade(SQLiteDatabase
    sqLiteDatabase, int oldVersion, int newVersion) {
        String sql = "DROP TABLE IF EXISTS " +
                        TABLE_NAME;
        sqLiteDatabase.execSQL(sql);
        onCreate(sqLiteDatabase);
    }
}
```

8.3.2 数据库操作

要想实现数据库“增、删、改、查”的操作，首先应新建一个类实例化上一节创建的 OrderDBHelper：

```
public OrderDao(Context context) {
    this.context = context;
    ordersDBHelper = new OrderDBHelper(context);
}
```

对实例化的 OrdersDBHelper 对象调用 getReadableDatabase()或 getWriteableDatabase()方法可以得到一个 SQLiteDatabase 类的实例，用于处理所有的数据操作方法。具体调用哪个方法取决于是否需要对数据库的内容进行修改：对于“增、删、改”这类对表内容变换的操作，需要调用 getWritableDatabase()，在执行的时候可以调用通用的 execSQL(String sql)方法或对应的操作 API——insert()、delete()、update()实现相应功能，而对“查”操作，需要调用 getReadableDatabase()，这时不能使用 execSQL 方法，而要使用 query()或 rawQuery()方法。下面对数据库的增、删、改、查操作进行进一步详细介绍。

1. 增加数据

在初始化数据时，由于一次性需要添加的数据比较多，通常使用 execSQL 方法；数据填充完成后如果需要插入数据，通常使用 insert(String table，String nullColumnHack，ContentValues values)方法，ContentValues 内部实现就是 HashMap，但是两者还是有差别的，ContenValues Key 只能是 String 类型，Value

只能存储 String、Int 等基本类型的数据，不能存储对象。

```
//初始化数据
db = ordersDBHelper.getWritableDatabase();
db.beginTransaction();

db.execSQL("insert into " + OrderDBHelper.TABLE NAME
+ " (Id, CustomName, OrderPrice, Country) values (1,
'Arc', 100, 'China')");
db.execSQL("insert into " + OrderDBHelper.TABLE_NAME
+ " (Id, CustomName, OrderPrice, Country) values (2,
'Bor', 200, 'USA')");
db.execSQL("insert into " + OrderDBHelper.TABLE NAME
+ " (Id, CustomName, OrderPrice, Country) values (3,
'Cut', 500, 'Japan')");
db.execSQL("insert into " + OrderDBHelper.TABLE NAME
+ " (Id, CustomName, OrderPrice, Country) values (4,
'Bor', 300, 'USA')");
db.execSQL("insert into " + OrderDBHelper.TABLE_NAME
+ " (Id, CustomName, OrderPrice, Country) values (5,
'Arc', 600, 'China')");
db.execSQL("insert into " + OrderDBHelper.TABLE NAME
+ " (Id, CustomName, OrderPrice, Country) values (6,
'Doom', 200, 'China')");

db.setTransactionSuccessful();

//插入数据
db = ordersDBHelper.getWritableDatabase();
db.beginTransaction();

// insert into Orders(Id, CustomName, OrderPrice,
// Country) values (7, "Jne", 700, "China");
ContentValues contentValues = new ContentValues();
contentValues.put("Id", 7);
contentValues.put("CustomName", "Jne");
contentValues.put("OrderPrice", 700);
contentValues.put("Country", "China");
```

```
db.insertOrThrow(OrderDBHelper.TABLE_NAME, null,
contentValues);

db.setTransactionSuccessful();
```

2. 删除数据

删除数据同样有两种方法：一种方法是使用通用的 execSQL 执行 SQL 语句，另一种是用 delete(String table,String whereClause,String[] whereArgs)方法，其中 whereClause 是删除条件，whereArgs 是删除条件值数组。与增加数据相同，对于需要修改数据的行为以事务处理。

```
db = ordersDBHelper.getWritableDatabase();
db.beginTransaction();

// delete from Orders where Id = 7
db.delete(OrderDBHelper.TABLE_NAME, "Id = ?", new
String[]{String.valueOf(7)});
db.setTransactionSuccessful();
```

删除表也可以使用 execSQL 方法来实现，删除数据库可以使用 deleteDatabase 方法：

```
db = ordersDBHelper.getWritableDatabase();
//删除表
db.execSQL("DROP TABLE  Orders");

//删除数据库
this.deleteDatabase("myTest.db");
```

3. 修改数据

修改数据同样有两种方法：一种方法是使用通用的 execSQL 执行 SQL 语句，另一种是调用 update(String table,ContentValues values,String whereClause, String[] whereArgs)。

```
db = ordersDBHelper.getWritableDatabase();
db.beginTransaction();

// update Orders set OrderPrice = 800 where Id = 6
```

```
ContentValues cv = new ContentValues();
cv.put("OrderPrice", 800);
db.update(OrderDBHelper.TABLE NAME,
        cv,
        "Id = ?",
        new String[]{String.valueOf(6)});
db.setTransactionSuccessful();
```

4. 查找数据

查找数据有两种方法：一是 public Cursor query(String table,String[] columns, String selection,String[] selectionArgs,String groupBy,String having,String orderBy, String limit)，二是 public Cursor rawQuery(String sql, String[] selectionArgs)。rawQuery 的写法类似上面的 execSQL，可以直接传入 SQL SELECT 语句，但如果查询是动态的，使用这个方法就会非常复杂。例如，当需要查询的列在程序编译的时候不能确定，这时使用 query() 方法会方便很多。query()方法中的参数如下：

table——表名称。

columns——列名称数组。

selection——条件字句，相当于 where。

selectionArgs——条件字句，参数数组。

groupBy——分组列。

having——分组条件。

orderBy——排序列。

limit——分页查询限制。

Cursor——返回值，相当于结果集 ResultSet。

query()方法返回的类型都是 Cursor，Cursor 是一个游标接口，指向每一条数据，提供了遍历查询结果的方法，Cursor 游标常用方法如表 8-1 所示。

表 8-1　Cursor 常用方法

方　法	说　明
move	以当前的位置为参考，将 Cursor 移动到指定的位置，成功返回 true，失败返回 false
moveToPosition	将 Cursor 移动到指定的位置，成功返回 true，失败返回 false
moveToNext	将 Cursor 向前移动一个位置，成功返回 true，失败返回 false
moveToLast	将 Cursor 向后移动一个位置，成功返回 true，失败返回 false

续表

方　　法	说　　明
movetoFirst	将 Cursor 移动到第一行，成功返回 true，失败返回 false
isBeforeFirst	返回 Cursor 是否指向第一项数据之前
isAfterLast	返回 Cursor 是否指向最后一项数据之后
isClosed	返回 Cursor 是否关闭
isFirst	返回 Cursor 是否指向第一项数据
isLast	返回 Cursor 是否指向最后一项数据
isNull	返回指定位置的值是否为 null
getCount	返回总的数据项数
getInt	返回当前行中指定的索引数据

【例 8-7】 查询用户 Bor 的信息

```
db = ordersDBHelper.getReadableDatabase();

// select * from Orders where CustomName = 'Bor'
cursor = db.query(OrderDBHelper.TABLE NAME,
        ORDER_COLUMNS,
        "CustomName = ?",
        new String[] {"Bor"},
        null, null, null);

if (cursor.getCount() > 0) {
        List<Order> orderList = new
        ArrayList<Order>(cursor.getCount());
        while (cursor.moveToNext()) {
        Order order = parseOrder(cursor);
        orderList.add(order);
    }
    return orderList;
}
```

8.4　ContentProvider 简介

ContentProvider 虽然与 Activity、Service、BroadcastReceiver 并列为 Android 四大组件，但如果不是特别开发一款与其他 App 有数据交互的应用，它的使用频

率远没有另外三者高。本节将从 ContentProvider 在框架中所充当的角色、ContentResolver 的使用，到 URI 的概念，再到数据共享的方法，一步步对 Content Provider 进行简要介绍。

8.4.1　ContentProvider 的角色

ContentProvider 为存储和获取数据提供统一的接口，可以在不同的应用程序之间共享数据。之所以使用 ContentProvider，主要有以下几个理由：

（1）ContentProvider 使用表的形式来组织数据，提供了对底层数据存储方式的抽象，使得开发者无须关心数据存储的细节。例如底层使用了 SQLite 数据库，在使用 ContentProvider 封装后，即使把数据库换成 MongoDB，也不会对上层数据使用层代码产生影响。

（2）Android 框架中的一些类需要 ContentProvider 类型数据。如果想让数据可以使用在如 SyncAdapter、Loader、CursorAdapter 等类上，就需要做一层 ContentProvider 封装。

（3）ContentProvider 为应用间的数据交互提供了一个安全的环境。它准许把自己的应用数据根据需求开放给其他应用进行增、删、改、查，而不用担心直接开放数据库权限而带来的安全问题。

8.4.2　ContentResolver

要对 ContentProvider 进行增、删、改、查的操作，需要借助一个新的类 ContentResolver，可以通过在所有继承 Context 的类中调用 getContentResolver() 来获得 ContentResolver。ContentResolver 类提供了与 ContentProvider 类相同签名的四个方法：public Uri insert(Uri uri, ContentValues values)、public int delete(Uri uri, String selection, String[] selectionArgs)、public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)、 public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)，分别用于实现对数据的“增、删、改、查”。

之所以不直接访问 Provider，而是在上面又加了一层 ContentResolver 来进行操作，是因为同一台手机中可能安装了很多含有 Provider 的应用。因此 Android 使用 ContentResolver 负责统一管理与不同 ContentProvider 间的操作，它使用 URI（Uniform Resource Identifier）来区别不同的 ContentProvider。

8.4.3 ContentProvider 中的 URI

ContentProvider 中的 URI 有固定格式，如图 8-1 所示。

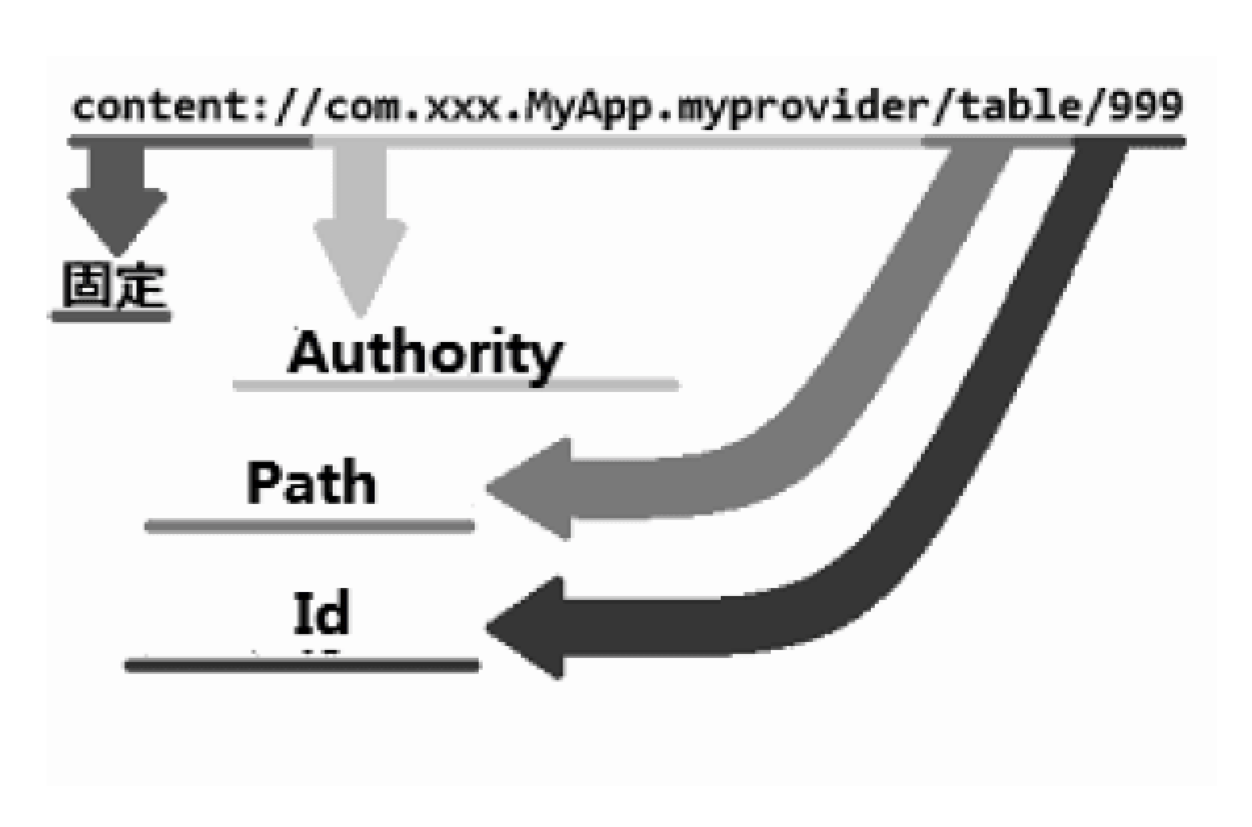


图 8-1 ContentProvider 的 URI

- Authority：主机名，唯一标识这个 ContentProvider；
- Path：表名，用于区分 ContentProvider 中不同的数据表；
- Id：Id 号，用于区别表中的不同数据。

如果要把一个字符串转换成 Uri，可以使用 Uri 类中的 parse()方法，如下：

```
Uri uri = Uri.parse("content://com.ljq.provider.personprovider/person")
```

因为 Uri 代表了要操作的数据，所以常常需要解析 Uri 并从中获取数据。Android 系统提供了两个用于操作 Uri 的工具类，分别为 UriMatcher 和 ContentUris。

UriMatcher 类用于匹配 Uri，它的用法如下：

【例 8-8】 UriMatcher 类使用实例

```
//常量 UriMatcher.NO_MATCH 表示不匹配任何路径的返回码
UriMatcher  sMatcher = new
UriMatcher(UriMatcher.NO_MATCH);
//如果 match()方法匹配
//content://com.ljq.provider.personprovider/person 路径，
//返回匹配码为 1
sMatcher.addURI("com.ljq.provider.personprovider",
"person", 1);

//如果 match()方法匹配
```

```
//content://com.ljq.provider.personprovider/person/2
//30 路径，返回匹配码为 2
sMatcher.addURI("com.ljq.provider.personprovider",
"person/#", 2);//#号为通配符
switch
(sMatcher.match(Uri.parse("content://com.ljq.provide
r.personprovider/person/10"))) {
   case 1
     break;
   case 2
     break;
   default://不匹配
     break;
}
```

上述代码首先把需要匹配的 Uri 路径全部注册，然后就可以使用 sMatcher.match(uri)方法对输入的 Uri 进行匹配，如果匹配成功就返回匹配码，匹配码是注册 Uri 时调用 addURI()方法传入的第三个参数。

ContentUris 类用于操作 Uri 路径后面的 ID 部分，它有两个比较实用的方法：第一个方法 withAppendedId(uri, id)用于为路径加上 ID 部分；第二个 parseId(uri)方法用于从路径中获取 ID 部分。

【例 8-9】 ContentUris 类使用实例

```
//添加 ID
Uri uri =
Uri.parse("content://com.ljq.provider.personprovider
/person");
Uri resultUri = ContentUris.withAppendedId(uri, 10);

//获取 ID
Uri uri =
Uri.parse("content://com.ljq.provider.personprovider
/person/10");
long personId = ContentUris.parseId(uri);
```

8.4.4 数据共享的实现

当应用需要通过 ContentProvider 对外共享数据时，首先需要继承

ContentProvider 类并重写下面方法：onCreate()、query(Uri, String[], String, String[], String)、insert(Uri, ContentValues)、update(Uri, ContentValues, String, String[])、delete(Uri, String, String[])和 getType(Uri)。

（1）public boolean onCreate()：该方法在 ContentProvider 创建后就会被调用，Android 开机后，ContentProvider 在其他应用第一次访问它时才会被创建。

（2）public Uri insert(Uri uri, ContentValues values)：该方法用于供外部应用向 ContentProvider 中添加数据。

（3）public int delete(Uri uri, String selection, String[] selectionArgs)：该方法用于供外部应用从 ContentProvider 中删除数据。

（4）public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)：该方法用于供外部应用更新 ContentProvider 中的数据。

（5）public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)：该方法用于供外部应用从 ContentProvider 中获取数据。

（6）public String getType(Uri uri)：该方法用于返回当前 Uri 所代表数据的 MIME 类型。如果操作的数据属于集合类型，那么 MIME 类型字符串应该以 vnd.android.cursor.dir/开头，例如，要得到所有 person 记录的 Uri 为 content://com.ljq.provider.personprovider/person，那么返回的 MIME 类型字符串应该为"vnd.android.cursor.dir/person"。如果要操作的数据属于非集合类型数据，那么 MIME 类型字符串应该以 vnd.android.cursor.item/开头，例如，得到 ID 为 10 的 person 记录，Uri 为 content://com.ljq.provider.personprovider/person/10，那么返回的 MIME 类型字符串为"vnd.android.cursor.item/person"。

由于 ContentProvider 是 Android 的四大组件之一，所以要在应用程序中使用它还需在 AndroidManifest.xml 中使用<provider>对该 ContentProvider 进行注册，为了能让其他应用找到该 ContentProvider，ContentProvider 采用了 authorities（主机名/域名）对它进行唯一标识。

```
<manifest...>
  <application android:icon="@drawable/icon"
                    android:label="@string/app name">
    <provider
        android:authorities="com.example.
                                contentprovidertest"
         android:name=".provider.TestProvider"
```

```
                android:exported="true">
        </provider>
    </application>
</manifest>
```

下面以 ContentProvider 最常封装的数据类型——数据库为例进行数据共享与权限管理的说明。

在自定义 ContentProvider 之前，首先使用上一小节的 SQLite 知识建立数据源，然后新建一个子类继承 ContentProvider 类，从而建立访问数据源的内容提供器。

【例 8-10】 数据共享示例

```
//新建数据源
public class TestDBHelper extends SQLiteOpenHelper{
    private static final int DATABASE VERSION = 1;
    private static final String DATABASE NAME =
    "test.db";

    public TestDBHelper(Context context) {
        super(context, DATABASE NAME, null,
              DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        final String SQL CREATE CONTACT TABLE =
                "CREATE TABLE " +
                TestContract.TestEntry.TABLE NAME
                + "( "
                + TestContract.TestEntry. ID
                + " TEXT PRIMARY KEY, "
                + TestContract.TestEntry.COLUMN_NAME
                + " TEXT NOT NULL );";
        db.execSQL(SQL_CREATE_CONTACT_TABLE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int
```

```
    oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " +
                TestContract.TestEntry.TABLE NAME);
        onCreate(db);
    }
}
```

下面建立访问数据源的内容提供器，为了便于内容提供器使用 Uri，先建立一个 Uri 组装类，然后建立匹配器并添加匹配规则，重写各种操作数据的方法，重写的步骤如下：对 Uri 进行匹配—>获取读或写数据库对象—>根据匹配结果利用 switch 语句判断该执行哪种操作—>返回结果。

【例 8-11】 数据共享示例

```
public class TestContract {

    protected static final String CONTENT_AUTHORITY
    = " com.example.contentprovidertest";
    protected static final Uri BASE_CONTENT_URI =
    Uri.parse("content://" + CONTENT AUTHORITY);

    protected static final String PATH TEST = "test";
    public static final class TestEntry implements
    BaseColumns {
        public static final Uri CONTENT URI =
              BASE CONTENT URI.buildUpon().
              appendPath(PATH TEST).build();
        protected static Uri buildUri(long id) {
              return
              ContentUris.withAppendedId(CONTENT URI,
              id);
        }

    protected static final String TABLE_NAME ="test";

    public static final String COLUMN_NAME = "name";
    }
}
```

```java

//建立访问数据源的内容提供器
public class TestProvider extends ContentProvider{
    private TestDBHelper mOpenHelper;

    @Override
    public boolean onCreate() {
        mOpenHelper = new TestDBHelper(getContext());
        return true;
    }

    @Nullable
    @Override
    public String getType(Uri uri) {
        return null;
    }

    @Nullable
    @Override
    public Cursor query(Uri uri, String[] projection,
    String selection, String[] selectionArgs, String
    sortOrder) {
        final SQLiteDatabase db =
                mOpenHelper.getReadableDatabase();
        Cursor cursor = null;
        switch ( buildUriMatcher().match(uri)) {
            case TEST:
                cursor = db.query(TestContract.
                            TestEntry.TABLE NAME,
                            projection, selection,
                            selectionArgs,sortOrder,
                            null, null);
                break;
        }
        return cursor;
    }

    @Nullable
```

```
    @Override
    public Uri insert(Uri uri, ContentValues values)
    {
        final SQLiteDatabase db =
                mOpenHelper.getWritableDatabase();
        Uri returnUri;
        long  id;
        switch ( buildUriMatcher().match(uri)) {
            case TEST:
                _id = db.insert(TestContract.
                        TestEntry.TABLE NAME, null,
                        values);
                if ( _id > 0 )
                        returnUri = TestContract.TestEntry.
                        buildUri(_id);
                else
                        throw new android.database.
                        SQLException("Failed to insert
                        row into " + uri);
                break;
            default:
                throw new android.database.
                SQLException("Unknown uri: " + uri);
        }
        return returnUri;
    }

    @Override
    public int delete(Uri uri, String selection,
    String[] selectionArgs) {
        return 0;
    }

    @Override
    public int update(Uri uri, ContentValues values,
    String selection, String[] selectionArgs) {
        return 0;
    }
```

```

    private final static int TEST = 100;

    static UriMatcher buildUriMatcher() {
        final UriMatcher matcher = new
               UriMatcher(UriMatcher.NO_MATCH);
        final String authority =
               TestContract.CONTENT_AUTHORITY;
        matcher.addURI(authority, TestContract.
                           PATH_TEST, TEST);
        return matcher;
    }
}
```

然后，就可以使用 getContentResolver()方法来对 ContentProvider 进行操作。

【例 8-12】 数据共享示例

```
public class MainActivity extends AppCompatActivity
 {
    @Override
    protected void onCreate(Bundle
    savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ContentValues contentValues = new
                                ContentValues();
        contentValues.put(TestContract.TestEntry.
                                     COLUMN_NAME, "Mike");
        contentValues.put(TestContract.TestEntry.
                  ID, System.currentTimeMillis());
        getContentResolver().insert(TestContract.
            TestEntry.CONTENT_URI, contentValues);

        Cursor cursor = getContentResolver().query
                          (TestContract.TestEntry.
                           CONTENT_URI,null, null,
                           null, null);
        try {
```

```

            Log.e("ContentProviderTest", "total data
                    number = " + cursor.getCount());
            cursor.moveToFirst();
            Log.e("ContentProviderTest", "total data
                    number = " + cursor.getString(1));
            } finally {
                cursor.close();
            }
        }
}
```

上面的例子是在本应用内对 ContentProvider 进行操作，当然开发者使用 ContentProvider 的目的是实现跨程序的数据访问。要让其他应用也可以访问本 Content Provider，首先要为应用程序添加 ContentProvider 的访问权限。

实现权限管理的一种方法是为此应用设置一个 android:sharedUserId，然后需要访问此数据的应用也设置同一个 sharedUserId，具有同样的 sharedUserId 的应用间可以共享数据。但这种方法不够安全，也无法做到对不同数据进行不同读写权限的管理，因此更好的方式还是使用 ContentProvider 中的数据共享规则。

共享数据会涉及以下几个重要标签：

android:exported——设置此 provider 是否可以被其他应用使用。

android:readPermission——该 provider 的读权限的标识。

android:writePermission——该 provider 的写权限标识。

android:permission provider——读写权限标识。

android:grantUriPermissions——临时权限标识，该标识为 true 时，意味着该 provider 下所有数据均可被临时使用；该标识为 false 时则反之，但可以通过设置 <grant-uri- permission>标签来指定哪些路径可以被临时使用。

最后，总结一下使用已定义好的 Content Provider 的整体流程：

（1）为应用程序添加 ContentProvider 的访问权限。

（2）通过 getContentResolver()方法得到 ContentResolver 对象。

（3）调用 ContentResolver 类的 query()方法查询数据，该方法会返回一个 Cursor 对象。

（4）对得到的 Cursor 对象进行分析，得到需要的数据。

（5）调用 Cursor 类的 close()方法将 Cursor 对象关闭。

习　题　8

1. 请尝试使用共享首选项保存用户自定义的配置信息，并在程序启动时自动启用这些自定义的配置信息。

2. 请完善习题 6.1 的学生信息录入系统，使得其具备把所录入学生的信息存储在本地并导出为 Excel 表的功能。

3. 建立一个 Content Provider 来共享上一题建立的数据库。

第9章 网 络 编 程

现在大多数手机都拥有五花八门的功能，但无论其功能多么丰富，手机就其本质而言终究还是一款通信终端。随着网络的不断更新换代，手机的网络功能也愈加丰富。20 世纪 80 年代第一代蜂窝移动通信系统（又称 1G 系统）大规模商用时，手机只有基本的语音通信功能。20 世纪 90 年代，随着 2G（TDMA、CDMA）及其后的 GPRS、EDGE 等技术的快速发展，手机才增加了数据服务功能。21 世纪第一个十年，3G 网络走上了历史的大舞台，并且随着 iPhone、Android 手机等智能移动设备的流行，一个颠覆的时代——移动互联网时代诞生了。如今，4G 网络也已大规模商用，移动设备上网速率已经超过了大部分家庭宽带速率。

作为移动互联网的“推动人”，Android 系统自然支持最新的网络制式，还支持 WiFi、NFC、蓝牙等常见的网络设备。

Android 的底层基于 Java，因此也支持 Java 语言自带的网络编程方式。除此之外，Android 还自带了开源项目 Apache HttpClient，作为 HTTP 扩展包。针对 WiFi、NFC、蓝牙，Android 还提供了单独的开发 API。

AndroidSDK 中的一些与网络有关的包如表 9-1 所示。

表 9-1　与网络有关的包

包　　名	主 要 功 能	最小的 API Level
Java.net	由 JDK 提供与联网有关的类，包括流和数据包、sockets、Internet 协议和常见 HTTP 处理	1
java.io	该包中的类可供其他 Java 包中提供的 Socket 和连接使用。它们还用于与本地文件（在与网络进行交互时会经常出现）的交互	1
org.apache	HTTP 扩展包，为 HTTP 通信提供精确控制和功能	1
android.net	除核心 java.net.* 类以外，包含额外的网络访问 Socket。该包包括 URI 类，后者频繁用于 Android 应用程序开发，而不仅仅是传统的联网方面	1
android.net.wifi	包含在 Android 平台上管理有关 WiFi（802.11 无线 Ethernet）所有方面的类	1

续表

包　　名	主 要 功 能	最小的 API Level
android.nfc	包含所有用来管理近场通信相关的功能类	9
android.net.http	包含处理 SSL 证书的类	1

这些包在功能上可能会有所重叠，但 Android 提供的网络编程方式可以归结为以下两种：基于 Socket 或基于 HTTP 的网络编程。

值得注意的是，为了让应用联网，需要在 mainfest 文件中添加以下权限：<uses-permission android:name="android.permission.INTERNET" />。

9.1　基于 Socket 的网络编程

在 TCP/IP 通信协议中，使用 IP 地址可以找到网络中特定的主机。但网络通信准确来说并不是两台计算机在通信，而是两台计算机上的进程在互相通信。因此还需要使用端口号以找到主机上对应的进程。Socket（套接字）正是 IP 地址与端口号的组合。Java 使用了 TCP/IP 套接字机制，实现了对 TCP、UDP 网络接口的封装，而不涉及上层协议。

TCP、UDP 的传输特性不同，因此适用于不同类型的网络应用。其中 TCP 面向连接，延时较长，能保证服务质量；UDP 无连接，数据包可能丢失或到达对端时顺序混乱，但延时短，效率高。因此 Java 提供了两种 Socket，分别对应 TCP、IP 协议。具体的常用类如表 9-2 所示。

表 9-2　与 Socket 有关的常用类

常　用　类	说　　明
Java.net.Socket	客户端连接使用的 TCP Socket
Java.net.DatagramSocket	客户端和服务端共同使用的 UDP Socket
Java.net.ServerSocket	服务端 TCP Socket 监听端口
Java.netInetAddress	IP 地址封装类
Java.net.DataGramPacket	通过 Datagram Socket 收发的数据包的封装类，包括数据和对端的 IP 地址、UDP 端口
Javax.net.SocketFactory	工厂类，控制客户端连接使用的 TCP Socket
Javax.net.ServerSocketFactory	工厂类，服务端 TCP Socket 监听端口
Javax.net.ssl.SSLSocketFactory	工厂类，SSL 客户端 Socket 构造器
Javax.net.ssl.SSLServerFactory	工厂类，SSL 服务端监听 Socket 构造器

9.1.1 UDP 套接字

Java 使用 DatagramSocket 类代表 UDP 协议的 Socket，DatagramSocket 本身不维护状态，不能产生 IO 流，它的唯一作用就是接收和发送数据报。此外，使用 DatagramPacket 来代表数据报，DatagramSocket 接收和发送的数据都是通过 DatagramPacket 对象完成的。当 Client/Server 程序使用 UDP 协议时，实际上并没有明显的服务器端和客户端，因为两方都需要先建立一个 DatagramSocket 对象，用来接收或发送数据报，然后使用 DatagramPacket 对象作为传输数据的载体。通常固定 IP 地址、固定端口的 DatagramSocket 对象所在的程序被称为服务器，因为该 DatagramSocket 可以主动接收客户端数据。

客户端使用 UDP 套接字主要执行以下三个步骤。

1. 创建 DatagramSocket 实例

DatagramSocket()：创建一个 DatagramSocket 实例，并将该对象绑定到本机默认 IP 地址、本机所有可用端口中随机选择的某个端口。

DatagramSocket(int prot)：创建一个 DatagramSocket 实例，并将该对象绑定到本机默认 IP 地址、指定端口。

DatagramSocket (int port, InetAddress laddr)：创建一个 DatagramSocket 实例，并将该对象绑定到指定 IP 地址、指定端口。

2. 发送和接收 DatagramPacket 实例

一旦得到了 DatagramSocket 实例之后，就可以通过如下两个方法来接收和发送数据。

receive(DatagramPacket p)：从该 DatagramSocket 中接收数据报。

send(DatagramPacket p)：从该 DatagramSocket 对象向外发送数据报。

3. 销毁套接字

使用 DatagramSocket 的实例方法 close()关闭 UDP 套接字。

服务器端使用 UDP 套接字主要执行以下三个步骤：

（1）创建一个指定了本地端口的 DatagramSocket 实例。

（2）使用 receive()方法接收一个来自客户端的 DatagramPacket 实例。鉴于 DatagramPacket 实例在客户端创建时就包含了客户端的地址，因此服务器端就知道消息来源以及消息需要回复到何处。DatagramPacket 类中获取消息来源的方法如下：

getAddress()——返回某台机器的 IP 地址，当程序准备发送此数据报时，该方法返回此数据报的目标机器的 IP 地址；当程序刚刚接收到一个数据报时，该方

法返回该数据报的发送主机的 IP 地址。

int getPort()——返回某台机器的端口，当程序准备发送此数据报时，该方法返回此数据报的目标机器的端口；当程序刚刚接收到一个数据报时，该方法返回该数据报的发送主机的端口。

SocketAddress getSocketAddress()——返回完整 SocketAddress，通常由 IP 地址和端口组成。当程序准备发送此数据报时，该方法返回此数据报的目标 SocketAddress；当程序刚刚接收到一个数据报时，该方法返回该数据报是源 SocketAddress。

（3）使用 DatagramSocket 类的 send()和 receive()方法来发送和接收 DatagramPacket 实例。

DatagramSocket 的实际应用，请在本节习题中完成。

9.1.2 TCP 套接字

Java 中基于 TCP 协议实现网络通信的类是适用于客户端的 Socket 类和适用于服务器端的 ServerSocket 类。JDK 使用它们封装了传输层上的 TCP 协议。

ServerSocket 对象会自动对其构造函数中传入的端口号进行监听，并在收到连接请求后，使用 ServerSocket.accept()方法返回一个连接的 Socket 对象。Socket 类在进行初始化时需要出入服务器端的 IP 地址和端口号，并返回连接到服务器端的一个 Socket 对象，如果连接失败，则返回异常，其通信过程如图 9-1 所示。

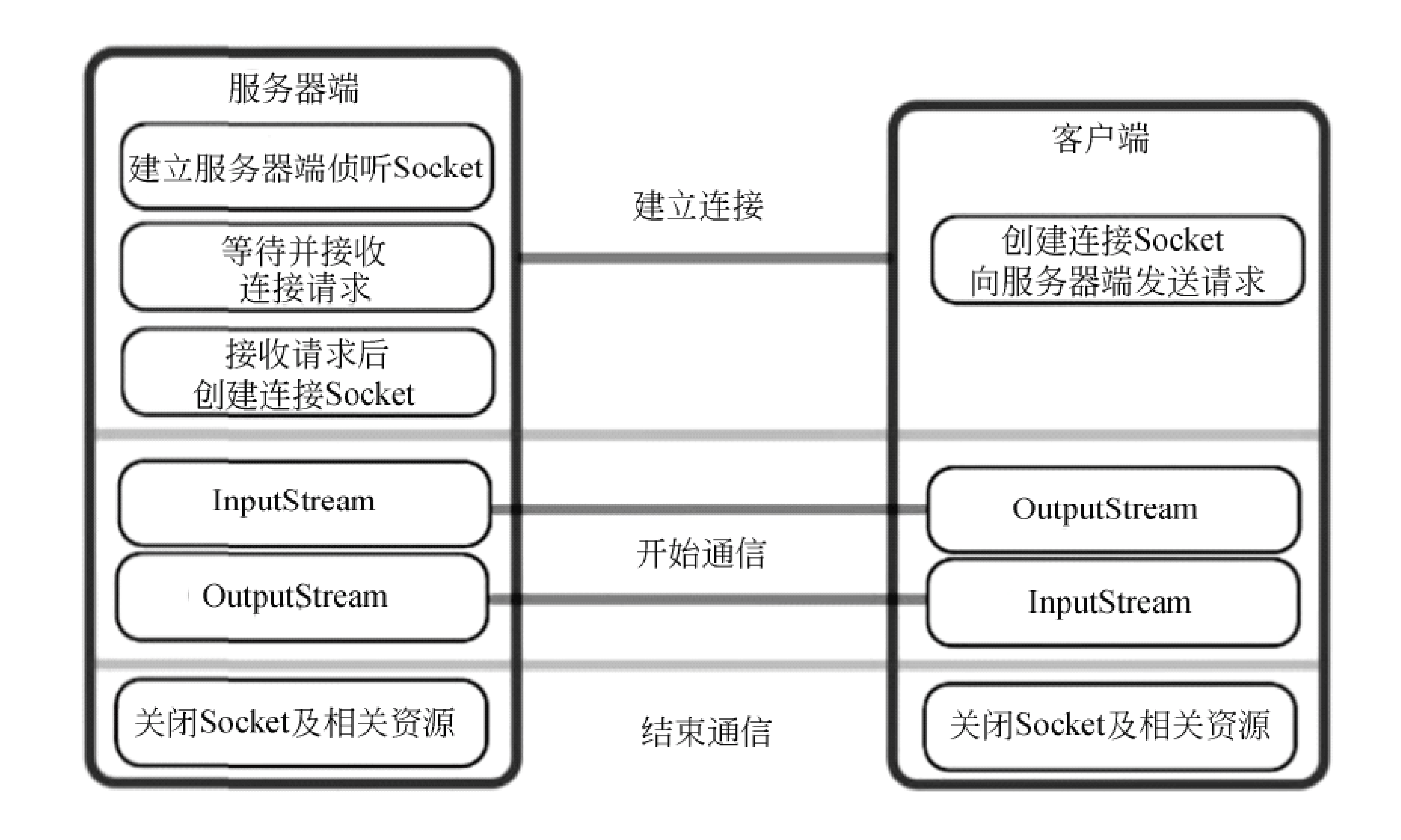


图 9-1　TCP 套接字通信过程

在客户端使用 TCP 套接字主要需要如下步骤：

（1）新建与指定服务器 IP 和端口号相连接的 Socket 对象，常用的构造方法为

```
Socket(String host, int port) throws UnknownHostException, IOException
```

其中 host 为服务器 IP 地址，port 为服务器进程对应的端口号。

（2）打开 Socket 的输入和输出流。

getInputStream()——获得输入流。输入流供应用程序要从流中取出数据时使用。

getOutStream()——获得输出流。输出流供应用程序需要对流进行数据写操作时使用。

（3）按照协议对 Socket 进行读写操作。

（4）关闭输入输出流和 Socket。

在服务器端使用 TCP 套接字主要需要如下步骤：

（1）创建 ServerSocket 对象，绑定监听端口。常用构造方法为

```
ServerSocket(int port)
ServerSocket(int port, int backlog)
ServerSocket(int port, int backlog, InetAddress bindAddr)
```

其中 port 为监听端口；backlog 为客户端连接请求的队列长度；bindAddr 为服务端绑定 IP（防止有多块网卡）。如果端口被占用或者没有权限使用某些端口会抛出 BindException 错误。如果设置端口为 0，则系统会自动为其分配一个端口。

（2）通过 accept()方法监听客户端请求。如果收到客户端连接请求，该方法会返回一个 Socket 对象。

（3）连接建立后，通过输入流读取客户端发送的请求信息。

（4）通过输出流向客户端发送相应信息

（5）关闭相关资源（但通常服务器的 ServerSocket 需要长期运行）。

【例 9-1】 示范 TCP 套接字的使用方法

打开 Android Studio，新建一个带空白 Activity 的项目，命名为 SocketDemo，作为客户端。本项目最终应该实现客户端能发送消息给服务器端，并且服务器端能返回原消息给原客户端。

修改 Activity 如下，添加一个 TextView 以显示发送的消息、一个 EditText 以编辑消息和一个发送按钮。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:layout_marginBottom="5dp"
    android:layout_marginTop="5dp">
    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="9">
            <TextView
                android:layout_width="match_parent"
                android:id="@+id/textView"
          android:textAppearance="@style/TextAppearance.AppCompat.Body1"
                android:layout_height="match_parent"
                android:layout_weight="9"
                android:lineSpacingExtra="10sp" />
    </ScrollView>
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="center_vertical">
        <EditText
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:id="@+id/editText"
            android:layout_weight="4" />
        <Button
            android:text="发送"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:id="@+id/button"
            android:layout_weight="1"
         />
```

```
    </LinearLayout>
</LinearLayout>
```

正常的即时通信软件的消息记录应该使用 ListView 展示的，但是考虑到篇幅有限，且为了突出 Socket 使用方法，本处简单地使用了 TextView 展示，并且使用了 ScrollView 使得 TextView 能上下滑动，以显示更多内容。

修改 MainActivity.java 如下：

```
package com.example.wb.socketdemo;
import …;
public class MainActivity extends AppCompatActivity implements
OnClickListener,
Runnable {
    TextView textView;
    Button button;
    EditText editText;
    Socket socket;
    private static final String HOST = "172.31.242.3";
    private static final int PORT = 2018;
    private BufferedReader in = null;
    private PrintWriter out = null;
    String str = "";//textView上显示的字符串
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //设置线程策略
        StrictMode.setThreadPolicy(new
              StrictMode.ThreadPolicy.Builder().detectNetwork()
              .penaltyLog().build());
        textView = (TextView)findViewById(R.id.textView);
        button = (Button)findViewById(R.id.button);
        editText = (EditText)findViewById(R.id.editText);
        button.setOnClickListener(this);
        client();
    }
    private void client(){
        try {
```

```
            socket = new Socket(HOST, PORT);
            in = new BufferedReader(new InputStreamReader(socket
                    .getInputStream()));//输入流
            out = new PrintWriter(new BufferedWriter(new
            OutputStreamWriter(
                    socket.getOutputStream())), true);//输出流
            new Thread(this).start();//启动线程，以监控服务器传来的消息
        } catch (Exception ex) {
         Toast.makeText(this,ex.getMessage(),Toast.LENGTH
         LONG).show();
            textView.setText(ex.getMessage());
        }
    }
    public Handler mHandler = new Handler() {
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            textView.setText(str);
        }
    };
    @Override
    public void run() {
        try {
            while (true) {
               if (socket.isConnected() && !socket.
               isInputShutdown()) {
                        String content;
                        if ((content = in.readLine()) != null) {
                            str += "服务器: "+content+"\n";
                           mHandler.sendMessage(mHandler
                           .obtainMessage());
                        }//end if
                }//end if
            }//end while
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    @Override
```

```
65      public void onClick(View v) {
66          if(v==button){
67              String msg = editText.getText().toString();
68              editText.setText("");
69              if (editText.getText().toString()!="" && socket
                .isConnected()&&
70                  !socket.isOutputShutdown()) {
71                      out.println(msg);
72                      str+="客户端: "+msg+"\n";
73                      textView.setText(str);
74              }//end if
75          }//end if
76      }//end onClick
77  }
```

其中第 19、20 行代码设置了线程策略，这是因为自从 Android 4.0 以后，不允许在主进程直接访问网络，以避免由于网络延时造成主线程堵塞。如果需要强制联网，需要加上如下代码：

```
StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().
detectNetwork().
penaltyLog().
build());
```

第 30～33 行，获取了 Socket 的输入流和输出流，同时为了方便操作和性能考虑，分别封装为 BufferedReader 和 PrintWriter。

第 40～45 行声明了一个 Handler，这是因为 Android 不允许在子线程中操控 UI，所以此处使用 Handler 机制，在 Handler 中操控 UI。

run()函数中使用了 while(true)死循环来不停地判断服务器是否发送了新消息。onClick()函数中使用 out.println(msg)实现了消息的发送。

服务器端使用 Eclipse+Java 开发，在 PC 上运行以模拟服务器。代码如下：

```
package src;
import …;
public class ServeSocketDemo {
    private static final int PORT = 2018;
    private List<Socket> mList = new ArrayList<Socket>();
```

```
                                                    //客户端 Socket 列表
        private ServerSocket server = null; //服务器 Socket
        private ExecutorService mExecutorService = null; //线程池
        public static void main(String[] args) {
            new ServeSocketDemo();
        }
        public ServeSocketDemo() {
            try {
                server = new ServerSocket(PORT);
                mExecutorService = Executors.newCachedThreadPool();
                System.out.println("开始监听");
                Socket client = null;
                while(true) {
                    client = server.accept();//如果有客户端连接成功,
                                                   //返回 Socket
                    mList.add(client);         //加入 Socket 列表
                    mExecutorService.execute(new Service(client));
                                               //创建新线程
                }
            }catch (Exception e) {
                e.printStackTrace();
            }
        }
        class Service implements Runnable {
                private Socket socket;
                private BufferedReader in = null;
                private PrintWriter out = null;
                private String msg = "";
                public Service(Socket socket) {
                    this.socket = socket;
                    try {
                        out = new PrintWriter(new BufferedWriter(
                        new OutputStreamWriter(socket
                        .getOutputStream())),true);
                        in = new BufferedReader(new
                        InputStreamReader(socket.getInputStream()));
                        //告诉客户端连接成功
                        msg="服务器与客户端"+this.socket
                        .getInetAddress()+"连接成功";
```

```
                        System.out.println(msg);
                        out.println(msg);
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
                @Override
                public void run() {
                    try {
                        while(true) {
                            if((!socket.isClosed()&&!socket
                            .isInputShutdown() &&
                            (msg = in.readLine())!= null)) {
      System.out.println("("+socket.getInetAddress()+")"+msg);
                              //如果 msg 为退出，就关闭连接
                                if(msg.equals("退出")) {
                                    String content = "与"
                                    +socket.getInetAddress()+"的连接断开";
                                    System.out.println(content);
                                    out.println(content);
                                    mList.remove(socket);
                                    in.close();
                                    out.close();
                                    socket.close();
                                    break;
                                } else {
                                out.println("("+socket
                                .getInetAddress()+")"+msg);
                                }
                            }
                        }
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
}
```

通常服务器不会只为一个客户端服务，因此需要使用多线程编程，此处采用了线程池的方式。第 13 行新建了一个 ServerSocket 对象。在第 17～21 行的 while(true)死循环中，服务器端不断监听是否有客户端请求连接。如果有请求，ServerSocket 会自动连接，并通过 accepted()函数返回 Socket。第 48 行重载了 run()函数，使得对应线程不断监听客户端是否发来消息。如果客户端发来消息为“退出”，就关闭对应 Socket 并释放资源，否则将客户端发送的消息原路返回给客户端。

客户端界面如图 9-2 所示，服务器端界面如图 9-3 所示。



图 9-2　客户端

```
Problems  @ Javadoc  Declaration  Console
ServeSocketDemo [Java Application] C:\Program Files\Java\jre1
开始监听
服务器与客户端/172.31.242.3连接成功
(/172.31.242.3)hello
(/172.31.242.3)this's the client
```

图 9-3 服务器端

9.2 基于 HTTP 的网络编程

HTTP（HyperText Transfer Protocol，超文本传输协议）是一种被广泛运用的基于 TCP 的网络协议。它是一个客户端和服务器端请求和应答的标准，用于从 WWW 服务器传输超文本到本地浏览器。通过使用 Web 浏览器、网络爬虫或者其他的工具，客户端发起一个到服务器上指定端口（默认端口为 80）的 HTTP 请求，服务器端响应之。使用 HTTP，可以减少网络传输，使浏览器更加高效。它不仅保证计算机正确快速地传输超文本文档，还能够确定传输文档中的哪一部分，以及哪部分内容首先显示（如文本先于图形）等。

HTTP 协议采用了请求/响应模型。客户端向服务器发送一个请求，请求头包含请求的方法、URL、协议版本、请求修饰符、客户信息和内容等。服务器以一个状态行作为响应，响应的内容包括消息协议的版本，状态码加上包含服务器信息、实体元信息以及可能的实体内容。常见的状态码有 200（OK）、400（Bad Request）、404（Not Found）等。

Http 协议最常用到的两种请求方式是 GET 方式和 POST 方式。

GET：从指定的资源请求数据。

POST：向指定的资源提交要被处理的数据。

Android 主要提供了两种类来实现基于 HTTP 的网络编程——HttpURLConnection 和 HttpClient。其中 HttpClient 来自 org.apache.http。对 Android 来说，在 2.3 版本之后建议使用 HttpURLConnection，之前建议使用 HttpClient。从 4.4 版本后，Android 使用 OkHttp 代替了 HttpClient。因此，这里只介绍 HttpURLConnection。

使用 HttpURLConnection 进行网络编程的基本流程如下：

（1）创建一个 URL 对象。

```
URL url = new URL(http://www.baidu.com);
```

（2）利用 HttpURLConnection 对象从网络中获取网页数据。

```
HttpURLConnection con = (HttpURLConnection) url.openConnection();
```

（3）设置连接超时。

```
con.setConnectTimeout(10*1000);
```

（4）对响应码进行判断。

```
if (con.getResponseCode() != 200 )
throw new Exception("请求 url 失败");
```

（5）得到网络返回的输入流。

```
InputStream in = con.getInputStream();
```

（6）从输入流获取内容。

（7）关闭连接和其他资源。

```
con.disconnect();
```

HttpURLConnection 默认的请求方式是 Get，如果想使用 Post，可如下设置：

```
con.setRequestMethod("POST");
```

【例 9-2】示范 HttpURLConnection 的使用方法。

打开 Android Studio，新建一个带空白 Activity 的项目，命名为 HttpURLConnectionDemo。

修改 activity_main.xml 如下：

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:layout_width="match_parent"
```

```
        android:layout height="match parent"
        android:orientation="vertical" >

        <TextView
            android:text="TextView"
            android:layout_width="match_parent"
            android:layout height="wrap content"
            android:id="@+id/textView" />
    </LinearLayout>
</ScrollView>
```

其中 ScrollView 是为了使 TextView 有足够空间显示文本内容。

修改 MainActivity 如下：

```
package com.example.wb.httpurlconnectiondemo;
import …
public class MainActivity extends AppCompatActivity {
    TextView textView;
    final int RESULT = 1,EXCEPTION = 2;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity main);
        textView = (TextView)findViewById(R.id.textView);
        new MyThread().start();
    }
    private class MyThread extends Thread{
        @Override
        public void run(){
            Message message = new Message();
            try{
                URL url = new URL("https://www.baidu.com");
                HttpURLConnection con =
                (HttpURLConnection)url.openConnection();
                con.setConnectTimeout(10*1000);
                con.setDoInput(true);     //允许输入流，即允许下载
                con.setDoOutput(true);    //允许输出流，即允许上传
                con.setUseCaches(false); //不使用缓冲
```

```
                con.setRequestMethod("GET"); //使用 get 请求
                if(con.getResponseCode()!=200)
                    throw new Exception(con.getResponseCode()+"");
                InputStream in = con.getInputStream();
                BufferedReader bin= new
                BufferedReader(new InputStreamReader(in));
                StringBuffer result = new StringBuffer();
                String line;
                while((line = bin.readLine())!=null){
                    result.append(line);
                }
                message.what = RESULT;
                message.obj = result.toString();
            }catch (Exception e){
                message.what = EXCEPTION;
                message.obj = e.toString();
            }finally {
               mHandler.sendMessage(message);
            }
        }
    }
    public Handler mHandler = new Handler() {
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            switch(msg.what){
                case RESULT:
                    textView.setText((String)msg.obj);
                    break;
                case EXCEPTION:
                    textView.setText("错误代码:"+msg.obj.toString());
                    break;
            }
        }
    };
}
```

由于 Android 默认不允许在主线程中进行联网操作，所以此处创建了一个线

程类 MyThread，也可以直接实现 Runable 接口。同时，由于默认不允许在子线程中更新 UI，所以此处使用了 Handler 机制。

最终运行截图如图 9-4 所示。

```
HttpURLConnection
<!DOCTYPE html><html><!--STATUS OK--><head><meta
name="referrer" content="always" /><meta charset='utf-8' /
><meta name="viewport" content="width=device-width,minimum-
scale=1.0,maximum-scale=1.0,user-scalable=no"/><meta http-
equiv="x-dns-prefetch-control" content="on"><link rel="dns-
prefetch" href="//m.baidu.com"/><link rel="shortcut icon"
href="https://gss0.bdstatic.com/5bd1bjqh_Q23odCf/static/
wiseindex/img/favicon.ico" type="image/x-icon"><link rel="apple-
touch-icon-precomposed" href="https://gss0.bdstatic.com/
5bd1bjqh_Q23odCf/static/wiseindex/img/screen_icon.png"/
><meta name="format-detection" content="telephone=no"/
><noscript><style type="text/css">#page{display:none;}</
style><meta http-equiv="refresh" content="0; URL=http://
m.baidu.com/?from=844b&amp;vit=fps&amp;pu=sz
%401321_480&amp;t_noscript=jump" /></
noscript><title>百度一下</title><script type="text/
javascript">window.onerror=function(m,b,n,l,k){var f="//
hpd.baidu.com/v.gif";var c={tid:"259",ct:"1",cst:"9",logFrom:"
mid_news",logInfo:"jsException",r:"l"+(Date.now()),logExtra:
{rta:"wise"}};var d={};d.message=m||"";d.url=b||"";d.line=n||"";c.
logExtra["excep"]=d;try{c.logExtra=JSON.stringify(c.logExtra)}
catch(j){}var g="";for(var h in c){if(c.hasOwnProperty(h))
{g+="&"+h+"="+encodeURIComponent(c[h])}}var a=f
+"?"+g.slice(1);new Image().src=a};</script><style type="text/
css" id='spa-base-style'>#search-card {display: none;}</
style><style type="text/css">@font-face {font-family:
'icons';src: url(https://gss0.bdstatic.com/5bd1bjqh_Q23odCf/
static/wiseindex/iconfont/iconfont_9c0cdb21.eot);src:
url(https://gss0.bdstatic.com/5bd1bjqh_Q23odCf/
static/wiseindex/iconfont/iconfont_9c0cdb21.eot#iefix)
format('embedded-opentype'),url(https://gss0.bdstatic.com/
5bd1bjqh_Q23odCf/static/wiseindex/iconfont/
iconfont_11a872b9.woff) format('woff'),url(https://
gss0.bdstatic.com/5bd1bjqh_Q23odCf/static/wiseindex/
iconfont/iconfont_0b0a971d.ttf) format('truetype'),url(https://
gss0.bdstatic.com/5bd1bjqh_Q23odCf/static/wiseindex/
iconfont/iconfont_b8b3267e.svg#iconfont) format('svg');font-
```

图 9-4　HttpURLConnection

9.3　WebView

事实上，Android 还提供了用于浏览网页的简易方法，比如使用默认的 Intent 调用浏览器打开网页。

```
Uri uri = Uri.parse("http://www.baidu.com");
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
startActivity(intent);
```

除此之外，还可以使用 WebView 控件浏览网页。WebView(网页视图)能加载显示网页，可以将其视为一个浏览器。它使用了 WebKit 渲染引擎加载显示网页。所有需要使用 Web 浏览功能的 Android 应用程序都要创建 WebView 视图对象显示和处理请求的网络资源。目前，WebKit 模块支持 HTTP、HTTPS、FTP 以及 JavaScript 请求。WebView 作为应用程序的 UI 接口，为用户提供了一系列的网页浏览和用户交互接口，客户程序通过这些接口访问 WebKit 核心代码。

WebView 的主要方法如表 9-3 所示。

表 9-3　WebView 的主要方法

WebView 的主要方法	说　明
loadUrl(String url)	加载对应的网页
goBack()	返回上一个页面
getSettings	获取 WebSettings
setWebViewClient(WebViewClient client)	设置 WebViewClient，使得 WebView 支持超链接响应等功能

WebSettings 的主要方法如表 9-4 所示。

表 9-4　WebSettings 的主要方法

WebSettings 的主要方法	说　明
setUseWideViewPort(true); setLoadWithOverviewMode(true);	这两个方法一起用，可以使网页适应手机屏幕
setSupportZoom(true)	支持缩放
setJavaScriptEnabled(true)	支持 JavaScript
setPluginsEnabled(true)	支持插件
setLayoutAlgorithm(LayoutAlgorithm.SINGLE_COLUMN);	支持内容重新布局

【例 9-3】 示范 WebView 的使用方法。

打开 Android Studio，新建一个带空白 Activity 的项目，命名为 WebViewDemo。

修改 activity_main.xml 如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```
    android:orientation="vertical">
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout height="wrap content"
        android:paddingLeft="16dp"
        android:paddingRight="16dp"
        android:paddingTop="16dp"
      >
        <EditText
            android:layout width="0dp"
            android:layout weight="10"
            android:layout_height="wrap_content"
            android:hint="网址"
            android:id="@+id/editText" />
        <Button
            android:text="GO"
            android:layout width="0dp"
            android:layout_weight="2"
            android:layout height="wrap content"
            android:id="@+id/button" />
    </LinearLayout>
    <WebView
        android:layout width="match parent"
        android:layout height="match parent"
        android:layout marginTop="16dp"
        android:id="@+id/webview" />
</LinearLayout>
```

其中 EditText 用来输入网址，输入网址后单击 Button 登录。

MainActivity.java 文件如下：

```
package com.example.wb.webviewdemo;
import…;
public class MainActivity extends AppCompatActivity {
    EditText editText;
    WebView webView;
    Button button;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        webView = (WebView)findViewById(R.id.webview);
        button =  (Button)findViewById(R.id.button);
        editText = (EditText)findViewById(R.id.editText);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                try{
                    String url = editText.getText().toString();
                    if(url!=null)
                        webView.loadUrl(url);
                }catch (Exception e){
Toast.makeText(getApplicationContext(),e.getMessage(),Toast.LENGTH_LONG);
                }
            }
        });
        webView.setWebViewClient(new WebViewClient(){
            @Override
            public boolean shouldOverrideUrlLoading(WebView view, String url) {
//返回值是 true 的时候控制去 WebView 打开，为 false 调用系统浏览器或第三方浏览器
                view.loadUrl(url);
                return true;
            }
        });
        WebSettings settings = webView.getSettings();
        settings.setJavaScriptEnabled(true);
        settings.setUseWideViewPort(true);
        settings.setLoadWithOverviewMode(true);
    }
    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_BACK && webView.
```

```
            canGoBack()) {
                //goBack()表示返回 WebView 的上一页面
                webView.goBack();
                return true;
            } else
                return super.onKeyDown(keyCode, event);
        }
    }
```

演示效果图如图 9-5 所示。



图 9-5　WebView

习　题　9

1. 完善习题 8.2 的学生录入系统，使得其具备与远程服务器同步学生信息的功能。

2. 请使用 WebView 组件实现一个较为完善的浏览器。

第 10 章　实战项目——2048 游戏

本章，我们将通过从零开始一步一步编写一款 2048 游戏，来对本书大部分知识点进行实践巩固。2048 是一个流行的益智游戏。游戏最开始时，界面会随机出现多个数字为 1 或 2 的方块。玩家可进行的操作为上下左右滑动屏幕，每滑动一次，所有的数字方块都会往滑动的方向靠拢。如果滑动前后方块的状态发生变化，系统会在空白的地方随机出现一个数字方块。相同数字的方块在靠拢、相撞时会相加，同时游戏的得分也会相应提高。

10.1　创建项目并编写界面样式

新建一个带空 Activity 的项目，命名为 2048。本游戏界面不需要进行 Activity 间的跳转，因此只有一个 Activity 即可。首先设计编写该 Activity 的 XML 样式文件，游戏界面主体应该是 m 行 n 列的方块构成的矩阵。可以用 TableLayout 作为矩阵的容器，每行用 TableRow 做容器。为了更灵活地控制矩阵的行数和列数，TableRow 和方块可采用代码动态添加的方式，activity_main.xml 的完整代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:orientation="vertical">

 <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
```

```
        android:layout_height="0dp"
        android:layout_weight="2"
        android:gravity="center">

    <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/current_score" />

    <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/currentScore"
            android:layout_marginRight="20dp" />

    <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/max_score" />

    <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/maxScore" />
    </LinearLayout>

    <TableLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="8"
        android:id="@+id/table">

    </TableLayout>
</LinearLayout>
```

其中，id 为 currentScore 的 TextView 用于在界面上显示当前分数，id 为 maxScore 的 TextView 用于在界面上显示历史最高分。

10.2 定义方块样式与行为

项目的难点在于定义方块的样式和行为，包括方块颜色、数值的变化、位置的移动和动画等。为便于自定义方块，可以自行创建一个 View，命名为 Cell。可

以利用 TextView 的 setText()功能来显示方块上的数字，因此令 Cell 继承自 TextView，并添加一个 int 类型的属性 num，用来标记该方块上显示的数字。

为提升用户体验，在游戏主体矩阵的状态发生变化时，新添加的数字方块在出现时应有动画效果。动画可以通过 XML 文件添加，也可以使用代码动态添加。这里通过在 Cell 中添加一个 AlphaAnimation 对象作为属性，来产生渐现动画效果。Cell 类的代码如下所示：

```
1   public class Cell extends TextView {
2       private int num = 0;
3       AlphaAnimation alphaAnimation = new AlphaAnimation(0f,1f);
4       public Cell(Context context) {
5           super(context);
6           init();
7       }
8       public Cell(Context context, AttributeSet attrs) {
9           super(context, attrs);
10          init();
11      }
12      public Cell(Context context,AttributeSet attrs,int defStyleAttr) {
13
14          super(context, attrs, defStyleAttr);
15          init();
16      }
17      private void init() {
18          TableRow.LayoutParams layout = new TableRow.LayoutParams(0,
19          TableRow.LayoutParams.WRAP_CONTENT, 1.0f);
20          layout.setMargins(10,10,10,10);
21          setLayoutParams(layout);
22          setGravity(Gravity.CENTER);
23          setTextColor(Color.WHITE);
24          alphaAnimation.setDuration(1000);
25          alphaAnimation.setFillAfter(true);
26      }
27  }
```

其中第 18～19 行，由于同一行的方块宽度应该是相同的，因此设置 LayoutParams 为：

```
new TableRow.LayoutParams(0,TableRow.LayoutParams.WRAP CONTENT,
1.0f);
```

该行代码等效于在 XML 中设置：

```
android:layout_width="0dp "
```

```
android:layout_height="wrap_content"
android:layout_weight="1"
```

为了更好的封装性，接下来，为属性 num 添加 get 和 set 方法。注意，当 num 为 0 时方块不显示文字。同时，为保证后续可以进行链式操作，set 方法返回 Cell 本身。此外，重载 setNum(int num, Boolean delay)方法，增加第二个参数用于控制是否展示渐现动画。在 Cell 类中增加如下代码：

```
…
public int getNum() {
    return num;
}

public Cell setNum(int num) {
    this.num = num;
    if (num == 0) {
        setText("");
    } else {
        setText(num + "");
    }
    return this;
}
public Cell setNum(int num, boolean delay) {
    if (delay) {
        startAnimation(alphaAnimation);
    }
    return setNum(num);
}
```

为了更好的美观性，希望方块的长宽严格相等，因此需要重写 onMeasure 函数。需要注意的是，这里不能直接使用 getWidth 和 getHeight 来获取宽高，因为此时 View 还没绘制，长宽为 0。OnMeasure 方法在 View 每次重绘的时候都会调用。它决定了 View 的大小。此处，由于已经设置了 weight 为 1，因此只需要令高度等于宽度即可。在 Cell 类中接着增加如下代码：

```
…
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec)
{
    super.onMeasure(widthMeasureSpec, widthMeasureSpec);
}
```

2048 游戏中每种数字对应的方块的颜色都是不同的，我们可以在 setNum 中

根据 num 的大小设置颜色，但更为规范的做法是重写 onDraw 方法。onDraw 方法与 onMeasure 方法类似，View 每次重绘的时候都会调用它。onDraw 解决的问题便是如何绘制这个 View。该方法参数为一个 Canvas 对象，通过对该对象的操作，我们可以生成各式各样的样式。此处只需简单地利用 setBackgroundColor 方法设置方块的背景色。因此，在 Cell 类中增加如下代码：

```
…
@Override
protected void onDraw(Canvas canvas) {
    setBackgroundColor(Constant.getColor(this.num));
    super.onDraw(canvas);
}
int getColor(int num) {
    if (num == 0) {
        return Color.argb(120,0xF1,0xDB,0x6C);
    }
    num *= 0xDB;
    return Color.argb(200,85,(int)(num*0.95)%255,(int)(num*
    0.98) % 255);
}
```

其中，getColor 是一个自定义函数，参数为方块的数字，用于返回数字对应的颜色。此函数中，可以手动设置对每一种数字返回特定的颜色。但这里为了简便，使用取模的方法来生成以蓝绿色为基色的颜色深浅不同的方块。这里 Color.argb 的参数可以自由调整，美观即可。

为了更好地规范项目中使用到的常量，新建 Constant 类，代码如下所示：

```
public final static int ROWS = 4;
public final static int COLUMNS = 4;
public final static String MAX_SCORE = "maxScore";
public final static int MIN_MOVE = 120;
public final static int MIN_VELOCITY = 0;
public enum Direction {
    up,
    right,
    down,
    left
}
public static int Directions[][] = {{-1, 0}, {0, 1}, {1, 0}, {0, -1}};
```

其中，ROWS 和 COLUMNS 可以任意改变，以控制游戏方块的数目。

MIN_MOVE 用于控制手势滑动时的最小移动位置，MIN_VELOCITY 控制最小移动速度。Direction 为枚举类型，表示上下左右四个方向。Directions 数组与 Direction 相对应，代表向相应方向移动一步时对应的行数和列数的变化。

10.3　编写 MainActivity

现在，开始实现游戏主体的 Activity。创建 MainActivity.java，代码如下所示：

```
public final static int ROWS = 4;
public class MainActivity extends AppCompatActivity implements
GestureDetector.OnGestureListener {
    private int rows = Constant.ROWS, columns = Constant.COLUMNS;
    private Cell cells[][] = new Cell[rows][columns];
    private TextView currentScoreElement, maxScoreElement;
    private int maxScore, currentScore;
    private GestureDetector detector;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        addCells();
        init();
    }
}
```

上面的代码中，rows 和 columns 代表行数和列数。cells 是由 rows 行 columns 列组成的自定义方块矩阵。maxScore 和 currentScore 分别为历史最高分和当前分，maxScoreElement 和 currentScoreElement 用来负责显示它们。为了检测手势，Activity 还要实现 GestureDetector.OnGestureListener 接口，并声明 GestureDetector 对象用来检测手势动作。

onCreate 方法中调用了 addCells 方法用于添加 rows 行 columns 列方块。下面实现 addCells 方法：首先获取 TableLayout，然后手动创建 TableRow 作为每行的容器并添加到 TableLayout 中。

```
…
private void addCells() {
    TableLayout table = (TableLayout) this.findViewById(R.id.table);
    for (int row = 0; row < rows; row++) {
        TableRow tableRow = new TableRow(this);
        tableRow.setLayoutParams(new
```

```
TableLayout.LayoutParams(ViewGroup.LayoutParams.MATCH PARENT,
ViewGroup.LayoutParams.WRAP CONTENT));
        table.addView(tableRow);
        addRow(tableRow, row);
    }
}
```

addRow 方法用于创建一行的方块，并将其添加到对应的 TableRow：

```
…
private void addRow(TableRow tableRow, int row) {
    for (int col = 0; col < columns; col++) {
        Cell textView = new Cell(this);
        cells[row][col] = textView;
        tableRow.addView(textView);
    }
}
```

接下来实现 init 方法。init 方法用于初始化变量，具体包括实例化手势监听器，获取显示分数的 TextView 并设置分数，其中最高分保存在 SharedPreferences 中，最后添加 columns 个数字方块。

```
…
private void init() {
    detector = new GestureDetector(this,this);
    maxScoreElement=(TextView) findViewById(R.id.maxScore);
    currentScoreElement=(TextView)
    findViewById(R.id.currentScore);
    SharedPreferences preferences =
    getPreferences(Context.MODE PRIVATE);
    maxScore = preferences.getInt(Constant.MAX SCORE, 0);
    maxScoreElement.setText(maxScore + ""); //获取历史最高分
    currentScore = 0;
    currentScoreElement.setText(currentScore + "");
    for (int i = 0; i < columns; i++) {
       addNum(false);
    }
}
```

addNum(booleandelay)方法的参数为 boolean 类型，控制是否显示渐现动画。本方法用于添加一个带数字的方块。它先将空方块的索引保存到一个数组里，然后从该数组里随意选出一个索引，再对该索引对应的空方块初始化：

```
…
private void addNum(boolean delay) {
    ArrayList<Integer> nullCells = new ArrayList<>();
    for(int i=0;i<rows; i++)
        for(int j=0; j<columns; j++){
            if(cells[i][j].getNum() == 0){
                nullCells.add(i*columns + j);
            }
        }
    if(nullCells.size() == 0){
        return;
    }
    Random random = new Random();
    int index = random.nextInt(nullCells.size());
    cells[nullCells.get(index)/rows][nullCells.get(index)%
    columns].setNum(random.nextInt(2) + 1, delay);
}
```

至此便完成了游戏页面的初始化，接下来就要考虑游戏的操作步骤。游戏的操作由 move 方法控制：

```
…
private void move(Direction direction) {
    String status = getCellsStatus();
    if(direction == Direction.left)
        moveLeft();
    else if (direction == Direction.right)
        moveRight();
    else if (direction == Direction.up)
        moveUp();
    else if (direction == Direction.down)
        moveDown();
    checkGameOver();
    if (status.equals(getCellsStatus())) {
        return;
    }
    addNum(true);
    checkGameOver();
}

private void moveUp() {
    for (int col = 0; col < columns; col++) {
        boolean merged = false;
        for (int row = 0; row < rows; row++) {
```

```
24              merged = moveCell(Direction.up, row, col, merged);
25          }
26      }
27  }
28
29  private void moveDown(){
30      for(int col = 0; col < columns; col++){
31          boolean merged = false;
32          for(int row = rows - 1; row >= 0; row--){
33              merged = moveCell(Direction.down, row, col, merged);
34          }
35      }
36  }
37
38  private void moveRight(){
39      for(int row = 0; row < rows; row++){
40          boolean merged = false;
41          for(int col = columns - 1; col >= 0; col--){
42              merged = moveCell(Direction.right, row, col, merged);
43          }
44      }
45  }
46
47  private void moveLeft(){
48      for(int row = 0; row < rows; row++){
49          boolean merged = false;
50          for(int col = 0; col < columns; col++){
51              merged = moveCell(Direction.left, row, col, merged);
52          }
53      }
54  }
```

move 方法根据上下左右，分别调用 moveLeft、moveRight、moveUp、moveDown 方法。这四个方法可以写在一起，但为了逻辑更清晰，此处将它们分开写。每次移动后需要检查游戏是否结束，如上面代码第 12 行所示。最后，如果移动前后方块的状态没有发生变化，则直接返回；否则，在空余位置随机添加一个数值为 1 或 2 的方块，并再次检查游戏是否结束。

对于 moveLeft、moveRight、moveUp、moveDown 四个方法，可以看到它们都是先移动其移动方向上靠前的方块，这样可以保证每次移动某个方块的时候，如果其前一个位置有方块，那么这个方块一定是已经处理好的。merged 用于标记上一个方块是否发生了合并。此标记是为了解决同一个方块在一个滑动中被多次

合并的问题。

接下来，我们实现 moveCell 函数。moveCell 函数用于移动某个方块，有四个参数分别表示方向、所处行、所处列和本次滑动前一个方块是否已经合并过。返回值表示本次调用是否发生了滑动。由前面的介绍可知，moveCell 被调用的时候，如果它前方有方块，那么这个方块一定是已经处理好了的。如果当前方块是空位置的或者位于移动方向的最前位置处（边际）的话，则可以直接跳过；否则如果前一个位置的方块没有合并，并且当前方块的数字和前一个方块的数字相等，就合并并且更新分数；如果上一个方块没有数字的话就往前移动一步。

```
…
private boolean moveCell(Direction direction, int row, int col, boolean merged) {
    boolean mergedHere = false;
    Cell cellCurrent = cells[row][col];
    if (cellCurrent.getNum() == 0)
        return mergedHere;
    int rowNext = row + Constant.Directions[direction.ordinal()][0];
    int colNext = col + Constant.Directions[direction.ordinal()][1];
    if (rowNext >= rows || colNext >= columns || rowNext < 0 || colNext < 0)
        return mergedHere;
    Cell cellNext = cells[rowNext][colNext];
    if (!merged && cellNext.getNum() == cellCurrent.getNum() ) {
        mergedHere = true;
        cellNext.setNum(cellNext.getNum() * 2);
        updateScore(cellNext.getNum() * 2);
        cellCurrent.setNum(0);
    } else if (cellNext.getNum() == 0) {
        cellNext.setNum(cellCurrent.getNum());
        cellCurrent.setNum(0);
        mergedHere = moveCell(direction, rowNext, colNext, mergedHere);
    }
    return mergedHere;
}
```

更新分数使用 updateScore 方法，参数为所增加的分数。如果当前分数大于历史最高分，则更新历史最高分，并保存到 SharedPreferences 中。SharedPreferences 可以直接读取，但存储需要借助 Editor 对象。

```
…
```

```
private void updateScore(int add) {
    currentScore += add;
    currentScoreElement.setText(currentScore + "");
    //如果当前分数大于历史最高分，更新历史最高分。
    if (currentScore > maxScore) {
        maxScore = currentScore;
        maxScoreElement.setText(maxScore + "");
        SharedPreferences preferences =
getPreferences(Context.MODE_PRIVATE);
        SharedPreferences.Editor editor=preferences.edit();
        editor.putInt(Constant.MAX_SCORE, maxScore);
        editor.commit();
    }
}
```

getCellsStatus 用于将所有方块的状态转换为一个字符串，以便比较某次滑动前后矩阵是否发生了变化。

```
…
private String getCellsStatus() {
    String s = "";
    for (int row = 0; row < rows; row++)
        for (int col = 0; col < columns; col++){
            s += (cells[row][col].getNum() + "/");
        }
    return s;
}
```

接下来实现 checkGameover 方法。每次移动结束都要检查游戏是否结束。如果当前矩阵存在没有数字的方块，或者有相邻且数字相等的方块，则游戏尚未结束，否则游戏结束，调用 gameOver 方法提示用户。

```
…
private void checkGameOver() {
    //如果有空方块，游戏尚未结束
    for (int row = 0; row < rows; row++)
        for (int col = 0; col < columns; col++) {
            if (cells[row][col].getNum() == 0) {
                return;
            }
        }
    //如果某列存在相邻的数字相等的方块，游戏尚未结束
    for (int col = 0; col < columns; col++)
        for (int row = 0; row < rows - 1; row++) {
```

```
        if (cells[row][col].getNum() == cells[row + 1][col].getNum()){
            return;
        }
    }
    //如果某行存在相邻的数字相等的方块，游戏尚未结束
    for (int row = 0; row < rows; row++) {
        for (int col = 0; col < columns - 1; col++) {
            if (cells[row][col].getNum() == cells[row][col +
1].getNum()) {
                return;
            }
        }
    }
    //游戏结束
    gameOver();
}

private void gameOver() {
    new AlertDialog.Builder(this).
            setTitle(R.string.app name).
            setMessage(R.string.game_over).
            setPositiveButton(R.string.sure, new
DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int which) {
                    dialog.dismiss();
              }
            }
        ).show();
}
```

至此，便完成了游戏的大部分功能，接下来要实现手势识别。我们需要实现 OnGestureListener 接口，并重写 Activity 的 onTouchEvent 方法，将事件交由 GestureDetector 对象处理。

```
…
@Override
public boolean onTouchEvent(MotionEvent event) {
    return detector.onTouchEvent(event);
}
```

实现 OnGestureListener 接口，需要重写 onDown、onShowPress、onSingleTapUp、onScroll、onLongPress、onFling 方法。

- onDown：用户轻触触摸屏时触发。

- onShowPress：用户轻触触摸屏，尚未松开或拖动时触发。
- onSingleTapUp：用户（轻触触摸屏后）松开时触发。
- onScroll：用户按下触摸屏，并拖动时触发。
- onLongPress：用户长按触摸屏时触发。
- onFling：用户按下触摸屏、快速移动后松开时触发。

考虑到 2048 游戏的使用场景，我们需要在 onFling 中实现监听。如果手势满足要求的话，则调用 move 函数。

```
…
@Override
public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,
float velocityY) {
    float minMove = Constant.MIN_MOVE;  //最小滑动距离
    float minVelocity = Constant.MIN_VELOCITY;  //最小滑动速度
    float beginX = e1.getX();
    float endX = e2.getX();
    float beginY = e1.getY();
    float endY = e2.getY();
    float x = Math.abs(endX - beginX);
    float y = Math.abs(endY - beginY);
    if (Math.abs(velocityX) <= minVelocity) {
        return false;
    }
    Direction direction = null;
    if(beginX - endX > minMove && x - y > minMove){       //左滑
        direction = Direction.left;
    }else if(endX - beginX > minMove && x - y > minMove){ //右滑
        direction = Direction.right;
    }else if(beginY - endY > minMove && y - x > minMove){ //上滑
       direction = Direction.up;
    }else if(endY - beginY > minMove && y - x > minMove){ //下滑
       direction = Direction.down;
    }
    if (direction != null) {
        move(direction);
    }
    return false;
}
```

最后运行本项目，游戏截图如图 10-1 和图 10-2 所示。



图 10-1　行数列数都为 5

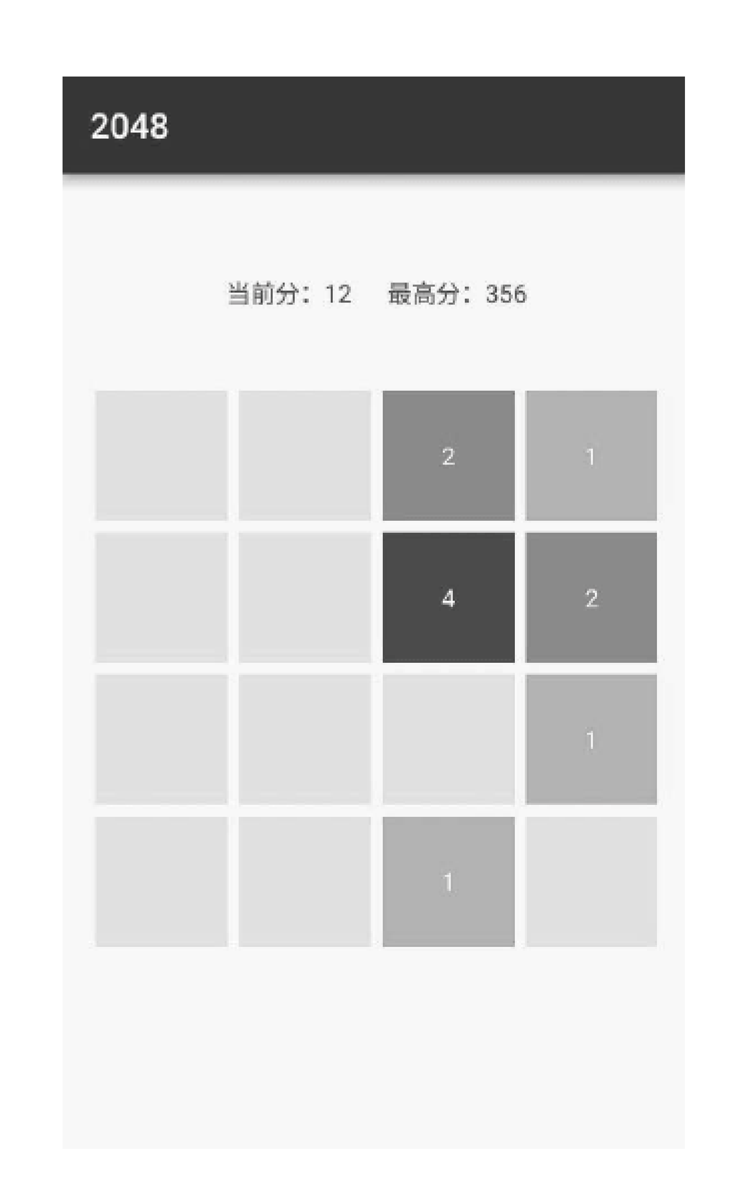


图 10-2　行数列数都为 4

至此，我们基本实现了 2048 游戏，当然本游戏还有很多地方可以优化，比如方块移动的音效等。有兴趣的读者不妨基于本项目实现更多功能。

附　录　A

2013 年 Google I/O 大会首次发布了 Android Studio IDE（Android 平台集成开发环境）。它基于 Intellij IDEA 开发环境，旨在取代 Eclipse 和 ADT（Android 开发者工具）为开发者提供更好的开发工具。相较 Eclipse，Android Studio 更为人性化，功能更为完善，还支持 Gradle 自动化构建工具。发布至今，已经取代 Eclipse 成为最主流的 Android IDE。

部署 Android 开发环境需要下载安装 JDK 和 Android SDK 以及 Android Studio。下面的默认环境为 Windows 系统。Mac 和部分 Linux 系统的安装方式大同小异，此处也可供参考。

1. 下载和安装 JDK

JDK 的全称是 Java SE Development Kit，也就是 Java 开发工具箱。SE 表示标准版。JDK 是 Java 的核心，包含了 Java 的运行环境(Java Runtime Environment)，大量 Java 工具和给开发者开发应用程序时调用的 Java 类库。下载时请注意物理机版本和 Java SDK 版本必须要保持一致，即：同为 64 位或者同为 32 位。JDK 的官方下载地址为 http://www.oracle. com/technetwork/java/javase/downloads/index.html，请尽量选择最新版本，以获得最佳体验。

下载到本地计算机后双击进行安装。JDK 的安装过程比较简单。在安装的时候只需要注意将 JDK 和 JRE 安装到同一个目录即可。JDK 默认安装成功后，会在系统目录下出现两个文件夹：一个代表 jdk，一个代表 jre，如图 A-1 所示。

图　A-1

2. 配置系统变量

为了配置 JDK 的系统变量环境，需要设置两个系统变量，分别是 JAVA_HOME

和 Path。

选择“系统属性”->“高级”->“环境变量”，在“系统变量”对话框中单击“新建”按钮，添加系统变量 JAVA_HOME，变量值为 JDK 的安装目录如图 A-2 所示。创建好后可以利用%JAVA_HOME%作为 JDK 安装目录的统一引用路径。

新建系统变量

变量名(N): JAVA_HOME

变量值(V): C:\Program Files\Java\jre1.8.0_111

浏览目录(D)... 浏览文件(F)... 确定 取消

图 A-2

PATH 属性已存在，可直接编辑，在原来变量后追加：;%JAVA_HOME%\bin，如图 A-3 所示。

编辑系统变量

变量名(N): Path

变量值(V): :\Program Files (x86)\Windows Kits\8.1\Windows Performance Toolkit\;%JAVA_HOME%\bin

浏览目录(D)... 浏览文件(F)... 确定 取消

图 A-3

3. 下载和安装安卓 SDK 以及 Android Studio

完整版的 Android Studio 包含 Android SDK，安装的时候也会顺带安装 SDK。下载链接为 https://developer.android.com/studio/index.html。国内用户可以到 http://www.android- studio.org 下载。下载时请注意操作系统是 32 位还是 64 位，如图 A-4 和图 A-5 所示。

下载完成后启动下载的.exe 文件。根据安装向导的指示安装 Android Studio 和所有所需的 SDK 工具即可，如图 A-6 所示。

Android Studio 是集成了 Android SDK 的，在安装的时候请选中 Android SDK，如图 A-7 所示。


Android Studio
FEATURES
USER GUIDE
Search
Back to Developers
DOWNLOAD
FEATURES
USER GUIDE
Android Studio
官方 Android IDE
Android Studio 提供用于为各类 Android 设备构建应用的最快速的工具。
利用世界一流的代码编辑、调试、性能工具、一套灵活的构建系统以及一套即时构建/部署系统，您可以专注于构建独特的高品质应用。
下载 ANDROID STUDIO
2.3 FOR WINDOWS (1,848 MB)
阅读文档
参阅发行说明

图　A-4

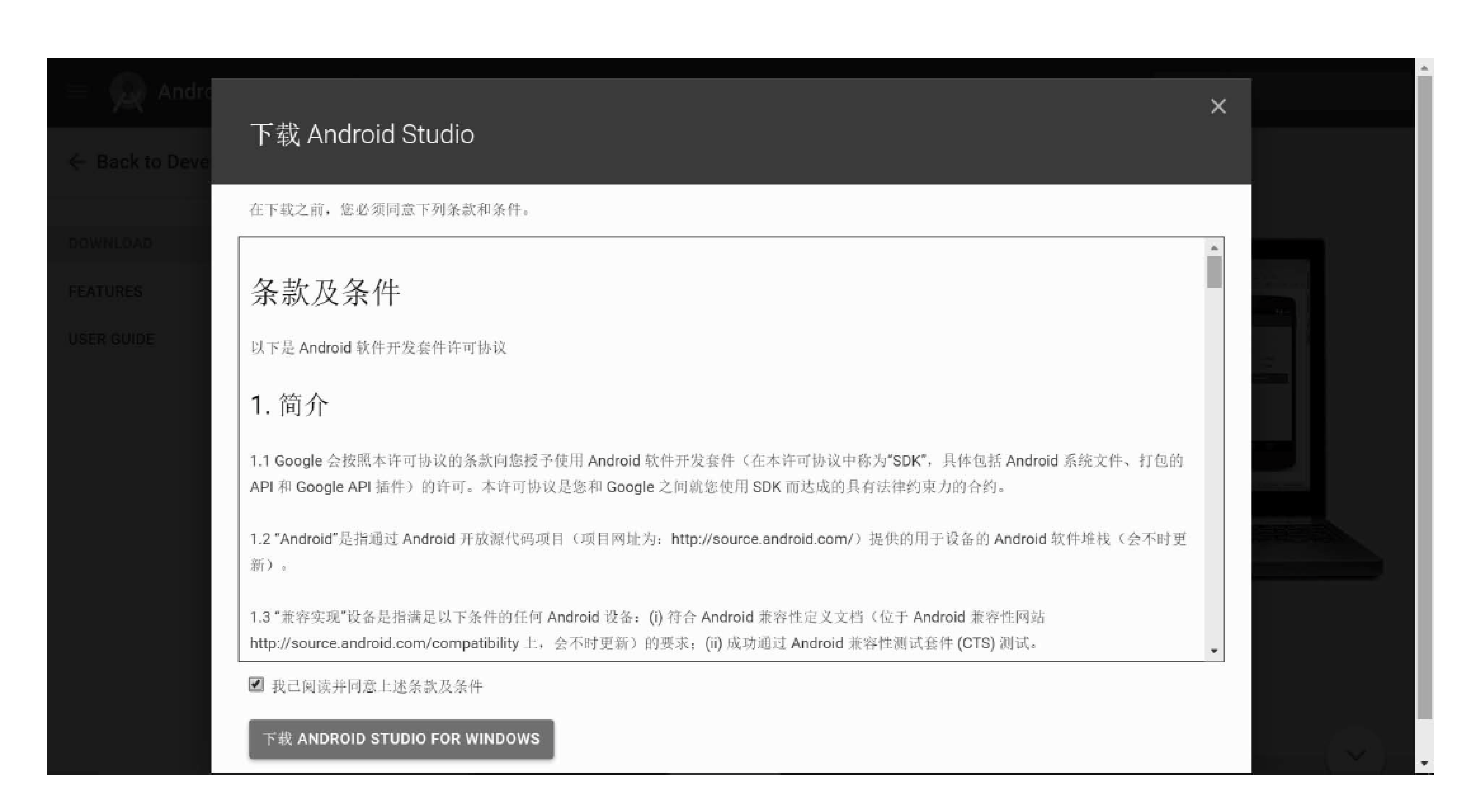

下载 Android Studio
在下载之前，您必须同意下列条款和条件。
条款及条件
以下是 Android 软件开发套件许可协议
1. 简介
1.1 Google 会按照本许可协议的条款向您授予使用 Android 软件开发套件（在本许可协议中称为"SDK"，具体包括 Android 系统文件、打包的 API 和 Google API 插件）的许可。本许可协议是您和 Google 之间就您使用 SDK 而达成的具有法律约束力的合约。
1.2 "Android"是指通过 Android 开放源代码项目（项目网址为：http://source.android.com/）提供的用于设备的 Android 软件堆栈（会不时更新）。
1.3 "兼容实现"设备是指满足以下条件的任何 Android 设备：(i) 符合 Android 兼容性定义文档（位于 Android 兼容性网站 http://source.android.com/compatibility 上，会不时更新）的要求；(ii) 成功通过 Android 兼容性测试套件 (CTS) 测试。
我已阅读并同意上述条款及条件
下载 ANDROID STUDIO FOR WINDOWS

图　A-5


Android Studio Setup
Welcome to Android Studio Setup
Setup will guide you through the installation of Android Studio.
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.
Click Next to continue.
Android Studio
< Back
Next >
Cancel

图　A-6


Android Studio Setup
Choose Components
Choose which features of Android Studio you want to install.
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.
Select components to install:
Android Studio
Android SDK
Android Virtual Device
Performance (Intel® HAX
Description
Position your mouse over a component to see its description.
全部选中
Space required: 3.8GB
< Back
Next >
Cancel

图　A-7

后续设置分别如图 A-8～图 A-11 所示。



图　A-8



图　A-9


Android Studio Setup
Configuration Settings
Emulator Setup
We have detected that your system can run the Android emulator in an accelerated performance mode.
Please set the maximum amount of RAM available for the Intel Hardware Accelerated Manager (HAXM) to use for all x86 emulator instances.
You can change these settings at any time. Please refer to the Intel HAXM Documentation for more information.
Recommended:
512 MB
Custom:
512
MB
* This value must be between 512 MB and 1 GB
Note: Setting aside a large memory reservation may cause other programs to run slowly when using the x86 Android emulator with HAXM.
< Back
Next >
Cancel

图　A-10


Android Studio Setup
Installing
Please wait while Android Studio is being installed.
Extract: guava-17.0.jar
Show details
正在安装
< Back
Next >
Cancel

图　A-11

等待一段时间后，安装完成，如图 A-12 和图 A-13 所示。

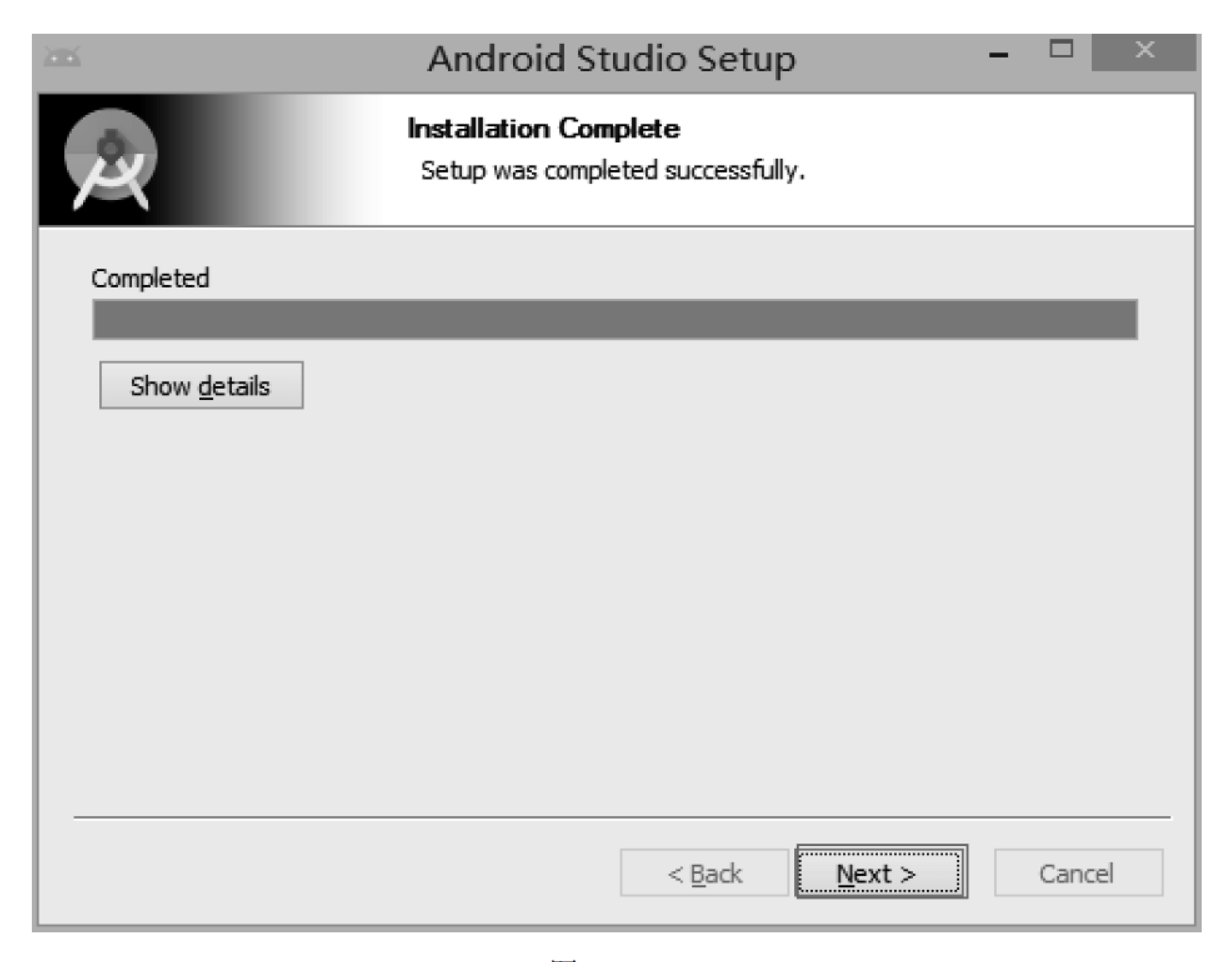


图 A-12



图 A-13

4. 启动 Android Studio

安装完成后，Android Studio 的启动页面如图 A-14 和图 A-15 所示。



图　A-14

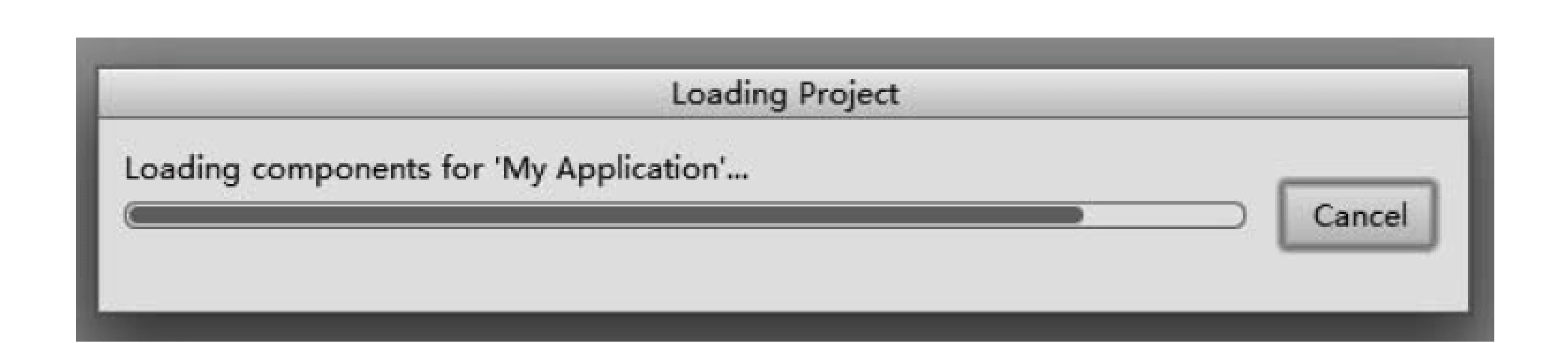


图　A-15

第一次启动 AndroidStudio 时，需要设置一下 SDK 的安装目录，因此会弹出如图 A-16 所示的对话框。



图　A-16

打开后的 Android Studio 页面如图 A-17 所示。

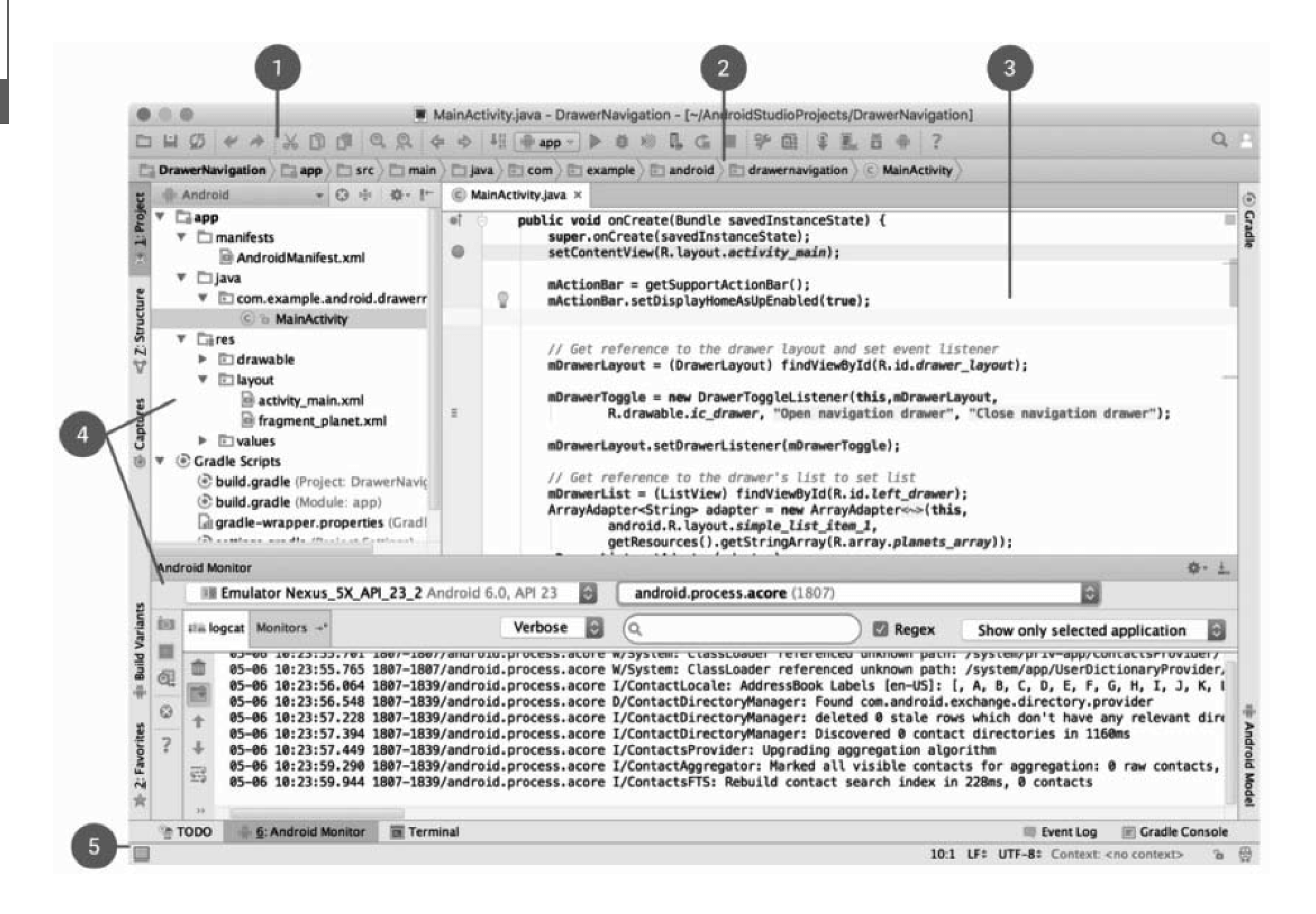


图　A-17

（1）工具栏提供执行各种操作的工具，包括运行应用和启动 Android 工具。

（2）导航栏可帮助您在项目中导航以及打开文件进行编辑。此区域提供 Project 窗口所示结构的精简视图。

（3）编辑器窗口是创建和修改代码的区域。编辑器可能因当前文件类型的不同而有所差异。例如，在查看布局文件时，编辑器显示布局编辑器。

（4）工具窗口提供对特定任务的访问，例如项目管理、搜索和版本控制等。可以展开和折叠这些窗口。

（5）状态栏显示项目和 IDE 本身的状态以及任何警告或消息。

参 考 文 献

[1] http://developer.android.com/guide/index.html.

[2] Meier R. Professional Android 4 application development[M]. Wrox Press Ltd. 2012.

[3] BillPhillips. Android 编程权威指南[M]. 2 版. 北京：人民邮电出版社，2016.

[4] 邓凡平. 深入理解 Android: 第 1 卷[M]. 北京：机械工业出版社，2011.

[5] 余志龙. Google Android SDK 开发范例大全[M]. 2 版. 北京：人民邮电出版社，2010.

[6] 任玉刚. Android 开发艺术探索[M]. 北京：电子工业出版社，2015.

[7] 柯元旦. Android 内核剖析[M]. 北京：电子工业出版社，2011.

[8] 郭霖. 第一行代码[M]. 北京：人民邮电出版社，2014.

[9] 黄彬华. Android 5.X App 开发实战[M]. 北京：清华大学出版社，2016.

[10] 张思民. Android 应用程序设计[M]. 北京：清华大学出版社，2013.

图书资源支持

感谢您一直以来对清华版图书的支持和爱护。为了配合本书的使用，本书提供配套的资源，有需求的读者请扫描下方的“书圈”微信公众号二维码，在图书专区下载，也可以拨打电话或发送电子邮件咨询。

如果您在使用本书的过程中遇到了什么问题，或者有相关图书出版计划，也请您发邮件告诉我们，以便我们更好地为您服务。

我们的联系方式：

地　　址：北京海淀区双清路学研大厦 A 座 707

邮　　编：100084

电　　话：010－62770175－4604

资源下载：http://www.tup.com.cn

电子邮件：weijj@tup.tsinghua.edu.cn

QQ：883604（请写明您的单位和姓名）



书圈

用微信扫一扫右边的二维码，即可关注清华大学出版社公众号“书圈”。